SAE-81-210

Semiannual Technical Summary

Distributed Sensor Networks

31 March 1981

Prepared for the Defense Advanced Research Projects Agency
under Electronic Systems Division Contract F19628-80-C-0002 by

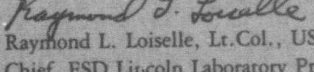
Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Lexington, Massachusetts

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Defense Advanced Research Projects Agency under Air Force Contract F19628-80-C-0002 (ARPA Order 3345).

This report may be reproduced to satisfy needs of U.S. Government agencies.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



Raymond L. Loiselle, Lt.Col., USAF
Chief, ESD Lincoln Laboratory Project Office

(12) LEVEL II

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

DISTRIBUTED SENSOR NETWORKS

SEMIANNUAL TECHNICAL SUMMARY REPORT TO THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

1 OCTOBER 1980 – 31 MARCH 1981

ISSUED 20 OCTOBER 1981

i/ii

LEXINGTON MASSACHUSETTS

ABSTRACT

This Semiannual Technical Summary reports work in the Distributed Sensor Networks program for the period 1 October 1980 through 31 March 1981. Status of the development and deployment of the DSN acoustic test bed and implementation of a real-time DSN capability using the test bed are reported. DSN communication requirements have been reviewed and a preliminary design has been completed for a communication channel structure and for protocols to provide the needed services. Simulation and analytical results have been obtained confirming the performance of the proposed distributed random-access scheme for local broadcast. Progress is reported on design studies directed toward an advanced nodal processor and software design and development efforts directed toward solving problems associated with distributed software systems.

CONTENTS

DISTRIBUTED SENSOR NETWORKS

I. INTRODUCTION AND SUMMARY

The Distributed Sensor Networks (DSN) program is aimed at developing and extending target surveillance and tracking technology in systems that employ multiple spatially distributed sensors and processing resources. Such a system would be made up of sensors, data bases, and processors distributed throughout an area and interconnected by an appropriate digital-data communication system. Surveillance and tracking of low-flying aircraft has been selected to develop and evaluate DSN concepts in the light of a specific system problem. A DSN test bed which will make use of multiple small acoustic arrays as sensors for low-flying aircraft is being developed and will be used to test and demonstrate DSN techniques and technology. This Semiannual Technical Summary (SATS) reports results for the period 1 October 1980 through 31 March 1981.

Progress in the design, deployment, and use of the acoustic DSN test bed at Lincoln Laboratory is reported in Sec. II. At present there are three operational nodes. These nodes have been used to collect data during aircraft flybys and extreme weather conditions. Two additional nodes are being constructed. Equipment is on order for a sixth node. Three vehicles with housings for the electronics of three DSN nodes are also on order. They will serve as deployable nodes for more flexible experimentation. Specifications have been written for an acoustically quieted generator which can supply power to the vehicle-mounted nodes.

During this past period we have also focused upon developing operational real-time test-bed capabilities, as opposed to data acquisition and off-line non-real-time processing of data. A decision was made to integrate MC68000 microcomputer hardware into the nodes to provide tracking services. The integration is completed for one of the nodes. We are now in the process of adapting a previously developed processing chain from raw data to target tracks for real-time operation. Data flow is controlled by a PDP-11/34

minicomputer in the node, signal processing is done primarily by a Floating Point Systems (FPS) AP-120B array processor, and tracking is assigned to the MC68000. The state of our efforts to achieve real-time operation is detailed in Sec. II. In addition, a node timer unit has been designed to provide a time standard for experimental purposes and a prototype has been built and checked out. Internodal communications in the test bed are now provided by telephone lines. Plans for using telephone lines in conjunction with a central microcomputer-based communication emulator have been formulated and are described.

A major part of our recent effort has been directed toward the communication aspects of a DSN. A separate DARPA-sponsored Communication Network Technology program at Lincoln Laboratory is developing a communication unit with the capability to measure range between radio units and with the capabilities needed for jam-resistant communications using pseudonoise spread-spectrum techniques with bit-by-bit code changing. When these units become available we plan to incorporate them into the DSN test bed to provide self-location capability and to provide communications between nodes. Our research in the communication area has assumed the availability of such radio units and has focused upon the eventual use of functionally similar units in DSNs of arbitrary size.

Section III reports the results of our efforts in the area of general DSN communications. Three kinds of DSN communication service have been emphasized. These are local one-radio-hop broadcast of sensor and target location data, multiple-hop distribution of information and commands over a region, and multiple-hop collection of information from a region. We have considered options for providing these services using radios with a nominal transmit-receive rate of 500 kbps and have designed a scheme which is based upon a slotted channel with slots assigned to the different kinds of service. The scheme for local broadcast is completely distributed but that for regional distribution and collection is hierarchical with certain access nodes performing at the top of the hierarchy. A topic for further investigation is how to provide more flexible and distributed services for regional information collection and distribution. We believe that features of the preliminary design can be

generalized and adapted to provide the required flexibility and decentralized operation.

The scheme for local broadcast of data uses randomized start times within each transmission slot and uses a listen-before-transmit protocol which allows it to make use of more than one out of every two slots with less than ten-percent collisions and very few totally suppressed transmissions. The small number of collisions and suppressed transmissions are acceptable due to the redundant nature of the information being exchanged. The scheme has several design parameters. Trade-offs and system performance as a function of these parameters have been investigated analytically and through simulation. The performance figures mentioned above represent acceptable values achieved in the simulations. The scheme for multihop regional distribution and collection of information exploits localized coordination of transmission times and the use of several different code spreading sequences for collision and congestion control.

Further investigation of an alternative time-division multiple access (TDMA) organization of the DSN communication channel has also been carried out and results are reported in Sec. III. An algorithm for assigning slots has been formulated and simulated. The algorithm correctly assigns slots but does not exploit the local nature of internode connectivity in a DSN so that it requires considerably more than the minimum number of slots, with a corresponding lack of usage of the communication channel.

Section IV describes advanced hardware and software concepts for DSN nodes as well as the development of related software tools. Options for the development of a multiple MC68000 DSN node processor are reviewed in the light of available Motorola computer products. The preliminary conclusion is that DSN requirements may best be met by a system making use of the MC68000 chip but with more memory than is now available on the commercial single-board systems offered by Motorola and with a packet-oriented bus interconnecting boards rather than a general-purpose processor and memory bus. Progress with the design and development of an advanced software system for distributed environments is also described. That software is to be a successor to the real-time kernel now supporting test-bed activities in the nodes and is

to be the basis for the software required for supporting the multiple MC68000 nodal computers. This advanced software effort addresses the issues of reliability, congestion, and throughput in distributed systems. To meet requirements with respect to these issues, an interprocess communication mechanism based upon queueable objects has been implemented in a limited way in the currently operating DSN test-bed nodes. Some variations upon the basic queueable object ideas are being introduced as we proceed with the detailed design of a general interprocess communication system for a distributed environment. A top-level view of two of these variations, "parameter" objects and "connectable" objects, is given in Sec. IV.

Section V reports on three miscellaneous items. These are an analysis of redundancy in a DSN, a review of our interactions with individuals and organizations outside Lincoln Laboratory, and a report of changes made to our software development and research computer facility.

II. DSN TEST-BED DEVELOPMENT

A. TEST-BED HARDWARE DEVELOPMENT AND DEPLOYMENT

At the present time three test-bed nodes are operational and the fourth and fifth nodes are being assembled. The PDP-11/34 computers and the analog-to-digital conversion equipment for the fourth and fifth nodes and the array processor for the fourth node have been diagnostically checked out. The balance of hardware for these two nodes is either on order or being fabricated in-house. Equipment for a sixth node is on order from the pertinent manufacturers or in the fabrication stage. It is planned that the fourth node will initially be located at the Lincoln Laboratory antenna test range on the eastern side of Hanscom Field. It will be installed as soon as the complete system is assembled. The equipment for the fifth and sixth nodes will be installed in mobile vehicles. A third mobile system will be implemented using the equipment installed at one of the three presently established fixed sites or possibly the equipment destined for the antenna range. This decision will be based on which of the four fixed nodes proved to be least useful due to location or background noise.

Initial tests with the third node at the Lincoln Flight Facility indicate that the location of the sensor array on the hangar roof is adversely impacted by activities in and around the hangar such that it is difficult to obtain good tracking data. The array is therefore being relocated approximately 600 ft south of the hangar. In the new location, the array will be near ground level (microphones elevated 4 ft) but will have an unobstructed field of view of more than 270° in azimuth. Longer signal cables (1000 ft) have been ordered from a local vendor and when they are received, the array will be moved to the new location.

As reported in the September 1980 SATS,† detailed specifications for vehicles to accommodate all necessary equipment for three mobile DSN nodes were prepared. Subsequently, contracts have been issued for the procurement

† Distributed Sensor Networks Semiannual Technical Summary, Lincoln Laboratory, M.I.T. (30 September 1980), DTIC AD-A103045.

of three vehicles. The basic vehicle is an 18,600-lb GVW truck with a special-purpose body housing mounted on the truck bed. The housing will accommodate all hardware for a DSN node. The equipment will be mounted in three 6-ft cabinet racks which will be shock-mounted from both the floor and overhead. The housing also will accommodate a desk or table for a control console, electronic test equipment, magnetic tape racks, and operating personnel. The housings also include the necessary air-conditioning and heating facilities.

Primary power for the mobile systems will be supplied by public utility sources when available. At other times, power will be provided by an on-board engine/generator (E/G) set. The E/G will be installed in a custom-designed sound attenuating enclosure and mounted on the truck bed at the rear of the body housing. The nature of the DSN system test requires low acoustic noise levels in the frequency band of 5 to 500 Hz. Detailed specifications for an E/G power system have been written and requests for bids have been distributed. Included in the power system specifications are strict acoustic specifications for the E/G enclosure and/or E/G modifications that will be necessary to reduce the noise level to an acceptable level for DSN system tests.

B. DEVELOPMENT OF REAL-TIME OPERATING CAPABILITIES

We are now proceeding with development and demonstration of real-time system operation in the test bed. The plan is to use previously developed location and signal-processing algorithms (SATS, 30 September 1980) but to implement them using cooperating processors as opposed to the sequential "batch" processing previously employed. This real-time implementation will involve inter- and intranode communications and associated communication techniques.

The plan for developing the real-time capabilities is divided into two major phases. In phase 1, the signal processing and single-site tracking will be done in each of the nodes, and the target locations will be computed centrally. In phase 2, each node will also perform multisite location. In this phase, the nodes will be interconnected by a communications environment emulator that will emulate the effect of digital radios using land-line links. Phase 1 will be carried out with two and three nodes and phase 2 will initially be carried out with three nodes and will subsequently be expanded to six.

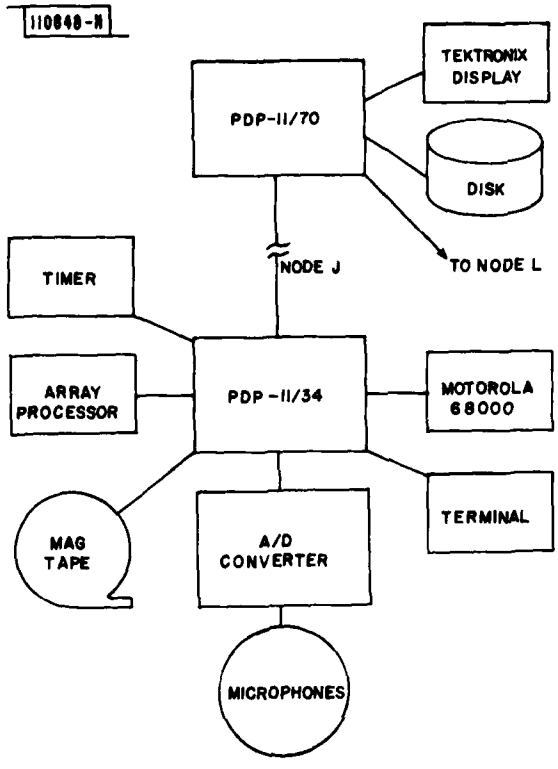


Fig. II-1. Phase 1 configuration for two-node real-time signal processing, location, and tracking.

The equipment configuration for phase 1 is shown in Fig. II-1. In this figure, two nodes (J and L) are connected to the central PDP-11/70 computer by 9600-baud telephone lines. Data flow within each node will be controlled by a PDP-11/34 processor. The data from the microphones will be digitized and passed to an array processor where they will be processed to determine the angles of arrival of peak sound energy. These "peaks" will be passed to an MC68000 processor, which will perform the clustering and single-site tracking functions. Resulting tracks will be sent to the PDP-11/70 computer that will log the data on disk files and perform multisite tracking. Tracks will be displayed on a Tektronix 4014 graphics display. The node configuration for phase 1 is essentially the configuration which has been discussed in previous

reports with the exception of the Motorola MC68000 processor and a timer board. Computation of the expected bus load on the PDP-11/34 and the expected demand of tracking algorithms indicated that there were inadequate bus cycles to run the tracking in the PDP-11/34 in real-time in combination with other required activities. Therefore, the MC68000 was added to provide processing required to support the tracking algorithms. Details of the purpose and design of the timer board are given in Sec. C below.

The configuration for phase 2 is shown in Fig. II-2. A second MC68000 will be added to accommodate both single-site and multisite tracking. In this

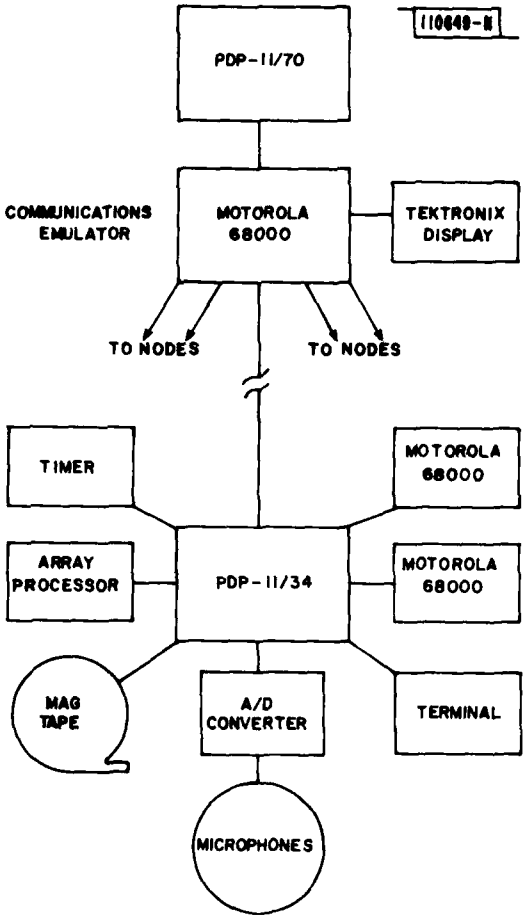


Fig. II-2. Phase 2 configuration for multiple-node real-time operation.

multiple processor configuration, single-site tracks from one MC68000 will be sent to the MC68000 location processor as well as to the communications emulator (another MC68000). The communications emulator will broadcast the single-site azimuth tracks to the other nodes. It will also receive the target locations which are broadcast by each node and appropriately route them. In addition, it will monitor selected tracks or locations on a graphic display. The phase 2 configuration will allow us to fully demonstrate cooperative processing between nodes. Further, by having the communications emulator lose packets or introduce errors, we can experiment with the effects of communication degradation on cooperative processes.

1. Phase 1 Hardware Status

One MC68000 processor has been procured and interfaced with the testbed node on J Building. It consists of a standard Motorola design module, to which we have added an additional 32 kB (kilobytes) of memory, as shown in Fig. II-3. All design module functions are on a single PC card. The processor runs at a 4-MHz clock rate and offers a wide range of instructions, combined with sixteen 32-bit internal registers. It has all the normal features of a microcomputer. Two chips provide two 8-bit parallel I/O interfaces and serial ASCII ports are each controlled by a single chip.

The bus on the design module is a hybrid between the old 6800 protocol and the new Versabus protocol that Motorola has specified as their standard for the MC68000. Future node enhancements will use Versamodule cards, which are based on the Versabus protocol. However the functional parts will be essentially identical to the configuration shown in Fig. II-3. ROM is used to store a down-line loader/debugging monitor and the hardware device table and other configuration parameters. Thus, application software can remain relatively configuration independent. Serial port 1 on the MC68000 is used for the operators terminal, and serial port 2 is used for connection to the host computer for down-loading programs. This configuration is now being used for program development. However, in the real-time operational environment the connections to the PDP-11/34 will be as shown in Fig. II-4. In this situation

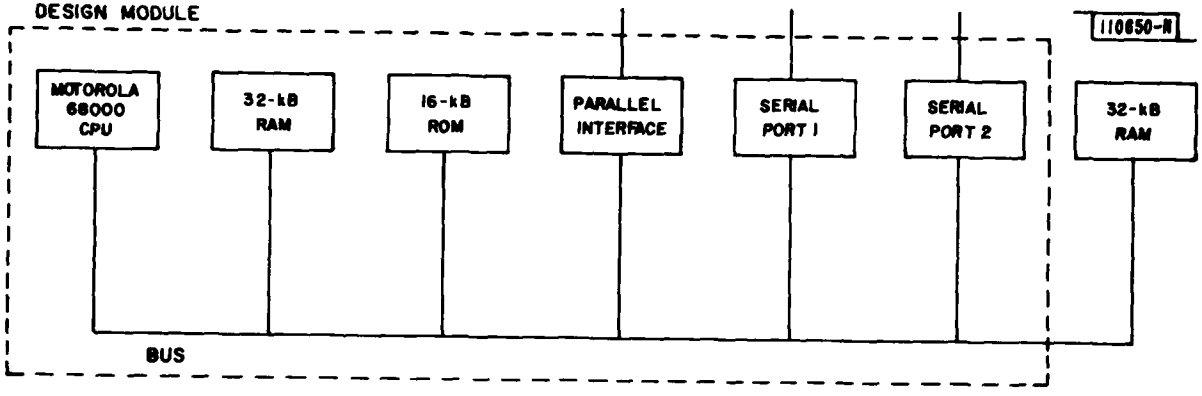


Fig. II-3. Phase 1 MC68000 processor configuration at a node.

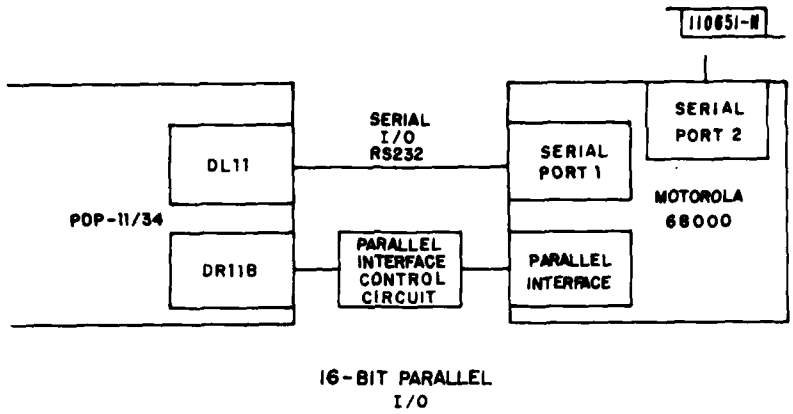


Fig. II-4. Interfaces between MC68000 and PDP-11/34 in test-bed nodes.

serial port 1 is connected to a DL11 interface on the PDP-11/34, such that the PDP-11/34 can control the MC68000. The MC68000 will initially be downloaded by means of a separate host computer connection to serial port 2.

The serial link does not provide enough bandwidth for real-time data transmission between the PDP-11/34 and the MC68000. A parallel interface has been developed to provide for necessary real-time data transfers. This interface uses a DR11B DMA interface on the PDP-11/34 side combined with two Programmable Interface Adapters operating in parallel programmed I/O mode in the MC68000. The interface control hardware has been designed and a control board has been built, installed, and tested on the first node. The control board provides for bi-directional 16-bit parallel transfers as well as providing the PDP-11/34 with the ability to reset and reinitialize the MC68000. The path from the PDP-11/34 to the MC68000 will be used to transfer data objects containing peak data. The reverse parallel path is currently unused. Azimuth tracks as well as diagnostic messages will be sent over the serial ASCII link to the PDP-11/34 which will send them to the PDP-11/70.

2. PDP-11/34 and Array Processor Software Status

Work is continuing on real-time signal-processing software for the DSN node. The goal is to implement FPS AP-120B array processor and

PDP-11/34 software that will perform DSN signal processing in real time and achieve the same processing performance as our off-line system, ADAP.

The departure point for the development of FPS software during this reporting period was previously developed software which ran on the Signal-Processing System (SPS) under UNIX on the PDP-11/70. SPS is an FPS code development system as described in our previous SATS (30 September 1980). The FPS code required improvement in two respects. First, it needed to be speeded up and second, some additional functional capabilities were required. To speed up processing, the signal-processing algorithms were carefully reviewed and "inner loops" (that is, the code executed most frequently during an analysis interval) have been replaced by carefully written microcode.

One important routine, which has been recoded to improve performance, is the routine which computes the acoustical power impinging upon the microphone array from a given "look" direction. This routine takes a complex (steering) vector, v, and a complex (cross spectral density) matrix, M, and evaluates a Hermitian form, using that matrix and vector. This routine must be executed 7680 times/2-s analysis interval and has been coded to make optimum use of the array processor. The finished routine consists of 44 words of array processor program source code. It executes in 52 μs. At the innermost loop, the machine is executing at the rate of 10 million floating-point operations per second. The average rate (which includes setup time) is in excess of 7 million floating-point operations per second for the Hermitian form evaluation. These timing calculations have been empirically confirmed.

It has also been possible to gain time in the area of system linkages between the various array processor routines. A vendor-supplied Program Development System produced very inefficient linkages and hand coding of linkages has been required to improve performance.

A peak picker has been added to the FPS software package. This routine searches a two-dimensional space of power values and returns peaks, i.e., local maxima. The algorithm compares the power value at a particular elevation and azimuth, i.e., a "look direction," with the power values in adjacent directions. If the power is greatest in the look direction, then that peak is added to a peak

list. This program has been written in the FPS assembly language since no higher-level routines were available.

Work has also continued on two PDP-11/34 signal-processing software modules for real-time operation. Design details were described in the SATS dated 30 September 1980. One module is the array processor server, the driver for the FPS processor. Its function is to control FPS program loading and execution as well as requests for reading and writing FPS data memory. The array processor server has been coded, tested, and released. The second module is the analysis server. It controls data buffer flow from the A/D converter to the array processor. It also manages the real-time signal processing in the array processor. The analysis server has been recoded to reflect changes to the underlying PDP-11 operating system (DAK2) and is ready for compiling.

Table II-1 provides details of the entire real-time signal-processing algorithm, system parameters, and the floating-point operations budget for a 2-s analysis interval. Note that "FIND POWER," which is the complex Hermitian form evaluation discussed above, constitutes more than half of the computation budget.

3. MC68000 Software Status

We are developing MC68000 software using software tools running under UNIX on the PDP-11/70 to produce absolute binary load modules that are down-line loaded over a serial ASCII link to the MC68000. We installed a C language compiler, symbolic assembler, and link editor that were developed by the Laboratory for Computer Science at M.I.T. We have added a down-line loader and a run-time library to the basic tools. The loader uses Motorola S-line format and operates in conjunction with the down-line loader ROM in the MC68000. The run-time library provides a subset of the services provided by system calls in a UNIX environment. Run-time library calls emulate their UNIX counterparts. This approach has proven worthwhile since conversion of existing application programs has been straightforward and we now have the capability to develop and test programs in the UNIX environment and subsequently compile them for use in the MC68000.

TABLE II-1

REAL-TIME SIGNAL-PROCESSING ALGORITHM OVERVIEW

I. ALGORITHM STRUCTURE

```
each analysis interval
        each block
                each channel
                        REFORMAT DATA
                        FAST FOURIER TRANSFORMS
                        COMPUTE AVERAGE POWER
                UPDATE CROSS SPECTRAL DENSITY MATRICES
        SELECT FREQUENCIES
        each frequency
                INVERT MATRICES
                each elevation
                        each azimuth
                                FORM STEERING VECTORS
                each elevation
                        each azimuth
                                COMPUTE POWER
                PICK PEAKS
```

II. SYSTEM PARAMETERS

C = NUMBER OF CHANNELS	9
A = NUMBER OF AZIMUTHS	120
B = NUMBER OF BLOCKS	8
L = BLOCK LENGTH	512
F = NUMBER OF FREQUENCIES	9
P = NUMBER OF PEAKS	100
E = NUMBER OF ELEVATIONS	8

III. FLOATING POINT OPERATIONS BUDGET

ALGORITHM	FLOATING POINT OPERATIONS	CURRENT SYSTEM
REFORMAT DATA	2*L*B*C	73728
FAST FOURIER TRANSFORM	2*L*LOG(L)*B*C	663552
AVERAGE POWER	2*L*B*C	73728
UPDATE CSDM	10*C*C*F*B	58320
SELECT FREQUENCIES	F*L	4608
INVERT MATRICES	8*C*C*C*F	52488
STEERING VECTORS	18*C*A*E*F	1399680
FIND POWER	(C/2) (9C-7)*A*E*F	2877120
PICK PEAKS	40*P	4000
FLOATING POINT OPERATIONS PER 2 SECOND INTERVAL		5207224

In addition, a list management package and graphics package have been converted to run on the MC68000. The former was necessary to provide support for the tracking algorithms and the latter was necessary to provide support for the interim test configuration shown in Fig. II-5. In this configuration,

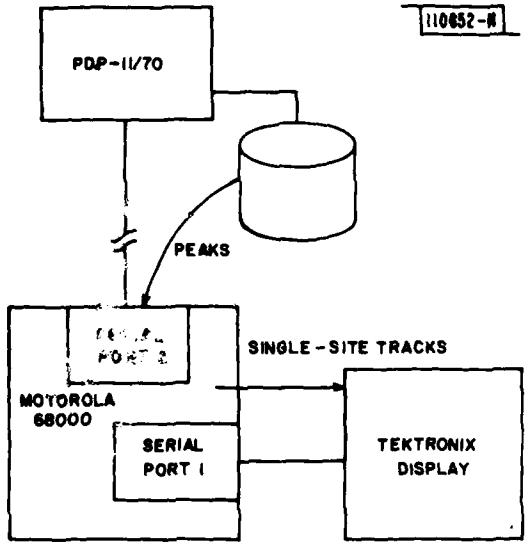


Fig. II-5. Interim test configuration.

a file of peaks is transmitted down the serial link to the MC68000 where they are used by the tracking programs. The results are then plotted directly on a Tektronix terminal to verify the correctness of the algorithms. Single-site azimuth clustering and tracking algorithms have now been converted to the MC68000 and azimuth tracks can now be plotted on a graphics terminal attached to the MC68000. Initial timing tests indicate that the processing of 2 s of peaks takes about 2 s of real time, but this includes time for transfer of the peaks to the MC68000 over the 9600-baud line. A more detailed timing analysis of the code is being done to determine how we might achieve a more comfortable real-time margin.

The next step will be to develop software to transmit the azimuth tracks back up to the PDP-11/70 where they will be spooled for a two-site tracking program.

C. NODE TIMER MODULE

A timer board is being developed that will use either a WWVB or GEOS satellite receiver to provide an experimental time standard for the test bed. It will be used to time stamp data and to eliminate the need to manually synchronize nodes. In addition it will provide a watchdog timer function and various system clocks, including a programmable rate clock for the analog-to-digital converter.

One part of the timer board is the interface to the WWVB or GEOS receiver. We have added to this interface the ability to read a switch-settable year value and a switch-settable node identification code. The interface includes provision for an interrupt from the external time-of-day source which serves to synchronize the local time counter to the external time to within 0.5 ms.

The timer includes a 16-bit 1.048576-MHz counter which can be read at any time to provide a high resolution measure of the amount of time taken by any process or the interval between any two events. One of the bits of this counter can be selected as the A/D sample clock. In addition, an 8-bit register is continually compared with the high-order 8 bits of the 16-bit counter to generate a processor interrupt at a software determined time. Using this feature, the software effectively extends the 16-bit counter to a 52-bit binary counter, which measures local time in seconds with 20-bit fractional accuracy. The interrupts are also used to schedule events at predetermined times. Considerable software simplicity results from the synchronization of the A/D, interval timer, and interval interrupt clocks, and from the use of a power of 2 frequency.

The watchdog timer uses the 16-Hz output of the main counter to count a resettable watchdog counter that triggers a computer reset when it overflows. The purpose of the watchdog timer is to allow a remote operator to regain control of the computer when the local program malfunctions and thereby fails to periodically reset the watchdog counter. This will allow completely reliable remote operation of DSN nodes. A watchdog time-out also generates a sonic alarm for the benefit of any local operator.

Lastly, 8-bit input and 8-bit output registers were included in the timer to eliminate the need for more than one DR11C interface to the PDP-11/34's.

The design of the DSN node timer is complete and the first unit has been constructed and successfully checked out with a PDP-11/34 computer. Checkout was accomplished using diagnostic test programs which exercised all functions of the timer except for the transfer of real time of day to the computer. This function was simulated by means of static switch inputs to the timer. Additional timer units are now being fabricated.

A WWVB synchronized time-of-day clock has been received and will be fully evaluated for performance and accuracy before additional units are ordered for the other five nodes.

D. TEST-BED COMMUNICATIONS

Prior to the incorporation of radio units with position location capability into the test bed, we are using 9600-baud modems and dedicated lines for internode communications. The maximum test-bed configuration will contain 6 separated nodes and therefore 15 lines and pairs of modems could be required for full communication connectivity. As an alternative we are designing a channel emulator unit which includes 6 serial ports for physical communication with nodes and which will emulate full connectivity by software means. The emulator will be at the center of a star with the nodes at the points. In addition to the six interfaces for the nodes, a seventh will connect the emulator to a PDP-11/70 host for down-line loading and an eighth will support an operator's terminal. The hardware configuration will be based upon a Motorola MC68000 Design Module with boards added to provide the extra serial ports and memory that will be required. Application software for the emulator will be written in C using the tools which have already been developed. We will also implement a simple process scheduler for the emulator. Additional routines to handle hardware interfaces and to provide interrupt driven I/O will also be written.

We have sized the test-bed communication requirements and have confirmed that they can be met by the emulator with 9600-baud lines between it and the nodes. Our present acoustic surveillance and tracking algorithms operate using

a 2-s sampling interval that imposes a 2-s cycle on communication requirements. The bulk of the traffic to be handled by the emulator and the 9600-baud lines will be "azimuth" and "track" packets. Each node will produce one of each type of packet per cycle and, for full connectivity in the six-node test bed, will receive five of each type per 2-s cycle. If a packet is considered as a single unit of traffic, then for half-duplex (HDUX) links the total load will be six units per second and for full-duplex (FDUX) links the load is five units receive and one unit transmit per second. Since HDUX is more limiting, the remainder of this sizing discussion is restricted to the case of HDUX operation.

We plan to use asynchronous modems which use ten bits to transmit each eight data bits so that the usable capacity of a link is $0.8 \times 9600 = 7680$ bps. For HDUX links these bits are distributed over six packets so that the available bits per packet are 1280. Suppose that each node reports on up to four target azimuths and four target tracks per cycle and that each report requires 240 data bits. Then an azimuth or track object will contain $4 \times 240 = 960$ data bits. This will fit into a 1280-bit packet with 320 bits left for packet headers and other overhead so that the 9600-baud HDUX links will handle the traffic for four reports of each kind per cycle.

To place the emulator in proper perspective, it should be pointed out that, as currently conceived, the radio and self-location units that will be developed for the test bed will consist of two subunits. One subunit (the radio unit) contains the SAW devices, transec, master timing, coder/decoder, and a small processor section. The other subunit (the main digital processor for the communications unit) is responsible for coordinating communications, computing self-location algorithms, and providing an interface with the node. The digital subunit also will have a serial port which can be connected to the emulator. This port can be a temporary replacement for the port to the radio unit thereby providing an alternative means for early test and evaluation in the test bed.

E. DATA COLLECTION

During this reporting period, we have made recordings of local area ambient noise including recordings made during a snowstorm. We also performed speaker calibration tests in the same storm. We expected the snow to have a

quieting effect on the environmental sound level, however, analysis on small amounts of the data proved to be inconclusive. More controlled experiments will be conducted in the future.

We also organized and executed a data-collection run with an A-10 jet aircraft flying a controlled linear track. These data have not yet been analyzed in any detail, except to verify the functioning of the nodes. This recording was done using nodes on L and J Buildings, and on the hangar roof of the flight facility. Recordings from this latter node were found to be very poor due to the reverberation of the hangar roof on which the array was mounted. For this reason this array is being relocated onto a grassy mound nearby.

III. DSN COMMUNICATIONS

An important aspect of a DSN is the communication network which interconnects the nodes. Progress with system design and research results related to the organization of the communication network are reported in this section.

For design and analysis purposes we have used the following radio characteristics. The radio will be half duplex and use pseudonoise spread-spectrum techniques. The pseudonoise (PN) chip sequence will be changed from bit-to-bit for reasons of contention control and jamming protection. Receivers will synchronize to a packet using the first few bits. A second packet arriving after synchronization will not hinder reception of the first packet because the second packet will be spread into white noise due to the mismatch between its chip sequence and that of the earlier packet. If the second packet arrives before synchronization of the first packet is completed, then neither packet will be received and the receiver will have no knowledge of this event. The minimum synchronization time will be on the order of 50 μs. We also assume that the receiver can quickly recognize if a packet is of interest and, if it is not, that the receiver can reset and search for new arriving packets. The time interval between onset of a packet to the recognition that it is not a packet of interest is called the "turnaround time." The bit rate to be provided by the radios will be in the 500- to 1000-kbps range and we have conservatively assumed the lower figure.

Sections A through E below report on an overall communication system design using the radios as described above, and Sec. F reports on a separate investigation into the use of a TDMA approach to DSN communication channel organization.

A. SYSTEM OVERVIEW

Each node in a DSN will have direct radio communications with a relatively small set of surrounding nodes. These surrounding nodes can be broken into two groups. The first "ring" of nodes around the node can be

termed as "near" neighbors. Near-neighbor pairs exchange data with each other in cooperative ventures such as multisite tracking. Communicating nodes beyond the first ring are "far" neighbors. Far neighbors are a source of contention for radio channel although they will not normally engage in cooperative tracking with the node. The radio units being developed will have adjustable power levels so that reliable communications can be provided with near neighbors while minimizing the number of far neighbors.

There are two different topological arrangements for a DSN. The first is a linear array conforming to a boundary such as a battleline. The second arrangement is a two-dimensional distribution over an area. We have focused our development and evaluation of communication strategies on the two-dimensional case. In that situation, a typical environment for a node might include 5 or 6 near neighbors and 10 to 20 far neighbors.

One essential DSN communication service is the exchange of sensor data between near neighbors. For the acoustic test bed these data will consist of "azimuthal" and "track" objects. Such information is exchanged on the order of once a second. The data exhibit considerable redundancy from second to second and more recent data are generally of more value than past data of the same kind. Given this situation, a highly reliable service with complex hand shaking and retransmissions is not needed for local distribution of sensor data but high throughput and known bounded delays are required. A random access broadcast protocol to provide the needed service is described in Sec. B and evaluated in subsequent sections.

At any point in a DSN users can eavesdrop on the local broadcast data to obtain local surveillance information. But in addition, there is a need to collect information from a larger area. The preliminary design presented in this report postulates the use of "access nodes" to provide for larger area access. The DSN is partitioned into regions with one active access node and up to about 50 total nodes in each region. Any node can function as an access node but the restriction to only one access node in an area does introduce a hierarchy into the system and results in design which is less distributed than we would like. In what follows, the large-area information distribution and

collection services are described as centered at these access nodes. Future research will be directed toward providing for more flexible and distributed access than that available through access nodes.

The existing hierarchical design provides for large-area information distribution and collection to be accomplished as follows. A multihop broadcast radiates outward from the access node to distribute information and provide coordination for report-back functions. As described in more detail in Sec. E, this outwardly radiating message provides for automatic route setup between a node and its access node as well as for distribution of commands and queries to the nodes. With the route back to the access node established, a means is provided to pass local information back to the access point. The mechanism for the implosion of local information inward to the access nodes uses local time-division and code-division multiplexing to avoid congestion near the access node.

The communication system must also provide for software maintenance functions such as down-line loading of programs to nodes and remote debugging of software. Two options are being considered. First, regional multihop broadcast from an access node can be used to distribute programs and debug commands. Acknowledgments and other status and debug information can be obtained by means of the area report-back function. Alternatively, receiver-directed transmissions could be used to implement point-to-point connections, with the participating nodes temporarily relieved of part of their other network functions.

The present DSN communication system design is based upon the use of a slotted channel. This will help to control contention as in the case of Slotted-Aloha vs pure Aloha. In addition, during normal operation a DSN can be viewed as a set of data collection and distribution programs that operate on a "cycle" whose duration is a function of the particular DSN. A nominal cycle time for an acoustic-sensor-based DSN with subsonic aircraft is on the order of 1 s. Based on this notion, we have divided time into intervals called cycles. We have also subdivided each cycle into 3 major sections corresponding to regional broadcast, local sensor data exchange, and report-back communications. The amounts allocated to each subdivision are system parameters.

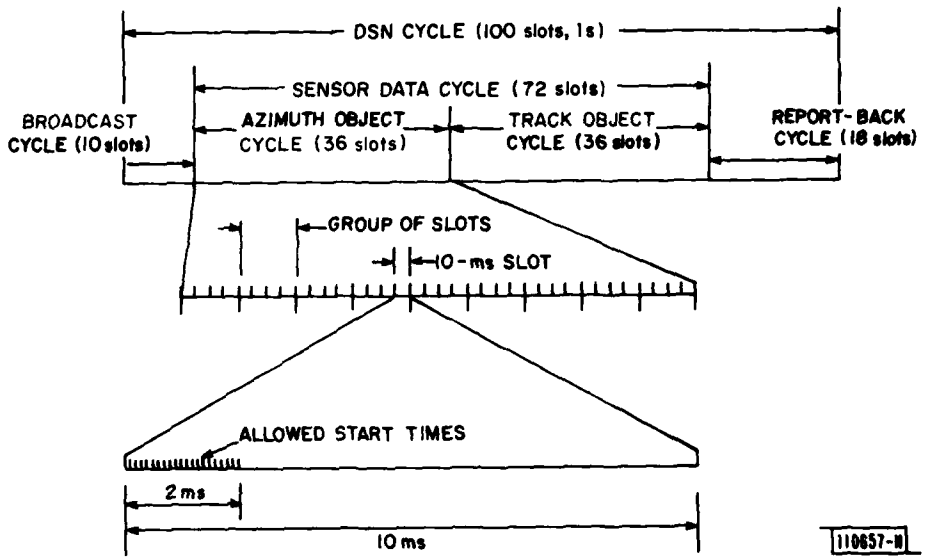


Fig. III-1. DSN communication channel organization. Numerical values shown are typical and may be adjusted during system operation or for different systems.

Figure III-1 schematically shows the overall channel organization as well as additional detail about the sensor data exchange part of the cycle. There are 100 10-ms slots/second. Ten slots are allocated for regional broadcast, 72 for local sensor data exchange, and 18 for report-back. The present design requires the alignment of the cycles at all the DSN nodes and the network-wide allocation of slots to similar communication functions. Future work will focus on investigating how we might improve the design and system performance by relaxing those requirements. The use of receiver-directed techniques for point-to-point communication during normal DSN operation is one simple example of a benefit that can be obtained by relaxing those assumptions.

B. RANDOM ACCESS PROTOCOL FOR LOCAL AREA BROADCAST

The communication load for sensor data exchange between near neighbors consists of one "azimuth" and one "track" object per cycle from each DSN node. Assuming each object reports on up to 15 possible targets and that

240 bits are used for each target, then each object consists of 3600 data bits. If 400 bits are allocated for preamble and header, this results in 4000-bit packets, requiring 8 ms to transmit at the 500-kbps rate.

Satisfactory exchange of these sensor data between near neighbors can be achieved as follows. The 8-ms data packets fit loosely into the 10-ms communication system slots. The 2 ms at the beginning of each slot is broken into equispaced intervals and the boundaries of these intervals are the only allowed start times for packet transmission. Transmitters randomly and independently schedule their transmissions to start on one of the allowed start times during the first 2 ms of the slot. A node which is already receiving a transmission from a near neighbor at the time of its own scheduled transmission aborts its own transmission and reschedules for a later time. All untransmitted data packets at the end of a cycle are discarded in favor of new data for the new cycle. The minimum spacing for the start times within the first 2 ms should be no less than 100 μs. This figure is the sum of a 50-μs receiver synchronization time and another 50 μs corresponding to the transit time over a receiver range of 15 km.

Following is a more detailed description of the protocol for providing local broadcast service. Specific numbers are used primarily for the purpose of exposition. Also, the protocol is given in terms of using 36 slots to provide for near-neighbor exchange of either "azimuth" or "track" objects with the understanding that another 36 slots can be used similarly and independently for the other kind of object. Assuming that up to 20 nodes can access the communication channel at any point, the nodes must cooperate to fit 20 packets into 36 slots with at most a few cases of lost packets due to multiple packets using the same slot. The distributed protocol presented here meets these requirements.

Assume that m = 20 possible packet start times are spaced over the first 2 ms of each slot. Also, arrange the slots in groups (assume s = 4 slots/group for a total of 9 groups). For the first group of 4 slots, let each node in the DSN independently pick a random number, k, between 1 and m*s*N where N is a number ≥ 1. The number k provides a schedule for the node

in this group. If a node selects k > ms it defers any attempted transmission to the next group in the cycle. Otherwise it is scheduled to attempt a transmission in slot S of the group and start time M in that slot according to the following rules. Given k, M is k modulo s and S is the integer quotient k/m. Thus for k = 52, s = 4, and m = 20, we get a node scheduling transmission at slot 52/20 = 2 and at start time 12 within that slot. The node listens for transmissions up to the actual time of its own transmission. It might hear a transmission originated by a neighbor at times 1 to 11 in the second slot. In that case it will wait for the next group in the cycle and select a new random number for scheduling transmission in that group. Once a node transmits in a cycle it does not participate any further until the next cycle.

Note that this protocol does not prevent all collisions or lost packets but, as we show below, the parameters can be adjusted to give satisfactory performance. Also note that if N is fixed during a cycle the collision performance will vary during the cycle since fewer nodes participate in later groups in the cycle. As will be seen below, it is desirable to have N > 1 for the first group and to gradually reduce it to one for the last group in a cycle.

C. SIMULATION OF RANDOM ACCESS PROTOCOL PERFORMANCE

A simulation has been implemented to evaluate the protocol proposed for local broadcast exchange of sensor data. For the simulation we assumed that nodes are embedded in a checkerboard grid. Each square contains one node with the location of the node in the square being randomly selected. For the results presented below the grid size was 5 km, any node within 7 km was taken as a near neighbor, and a node at ranges from 7 to 16 km was taken as a far neighbor.

The statistics to be obtained and evaluated are the following:

(1) "NEAR" packets lost through collisions. Since these are the primary information packets the minimization of the near-packet loss rate is a major goal.

(2) "FAR" packets lost through collisions. These are normally interfering packets and their loss is of no

consequence. However, there may be circumstances that a number of near nodes are inoperable and it may be useful to redefine one or more of the far nodes as replacements for coordinated tracking purposes.

(3) Transmission packets inhibited. With a low probability it is possible that a transmission from the center node may be inhibited for all the groups in a cycle. It seems clear that this should be minimized since each total inhibition of a transmission results in a loss of 5 or 6 lost near packets.

The simulations all assumed a 50-μs synchronization acquisition time and that a far packet arriving at least 50 μs after a near packet would not be detected or otherwise interfere with the continued reception of the near packet. It was also assumed that no members of a set of packets arriving within a 50-μs interval will be correctly received. The simulations used a turnaround time of 200 μs to reject a far packet and to reset to search for near packets. The impact of this feature will be investigated further although we do not believe it is significant.

Table III-1 shows results for a set of simulations with N = 1. In each group the nodes select a number from 1 to ms, where m is the number of possible start times at the beginning of a slot and s is the slots per group. Each node which has not yet transmitted in a cycle is thus forced to schedule a transmission. The first test in Table III-1 portrays the situation in which each node randomly selects one of the 36 slots in which to transmit and does not utilize a channel-sensing strategy since there is only one starting location per slot. The near-packet collision rate is about 40 percent. The best strategy in the table (No. 6) appears to be to divide the cycle into two groups. This gives a near-packet loss of 10.4 percent with a < 1-percent transmission inhibition. In case 7, there is a lower near-packet loss but since there is no opportunity for retransmission the inhibited transmissions rise to 6.4 percent.

Other runs with fixed values of N other than unity did not lead to any improved performance. But one modification which did improve performance

TABLE III-1

SIMULATION RESULTS WHEN EACH NODE WHICH HAS NOT YET TRANSMITTED IS FORCED TO SCHEDULE A TRANSMISSION IN EACH GROUP (N = 1). AVERAGE NUMBER OF NEAR NEIGHBORS = 5; AVERAGE NUMBER OF FAR NEIGHBORS = 13.

Test No.	m Starting Points in Slot	s Slots/ Group	Percent Far-Packet Collisions	Percent Near-Packet Collisions	Percent Inhibited Transmissions
1	1	36	41.9	40.3	0
2	10 (200 μs)	4	46.7	29.2	<1.0
3	20 (100 μs)	4	33.0	19.6	<1.0
4	20	6	27.4	18.4	<1.0
5	20	12	18.3	12.0	<1.0
6	20	18	12.6	10.4	<1.0
7	20	36	9.1	6.4	6.4

was to let $N > 1$ at the start of a cycle to reduce the fraction of scheduled transmissions when the number of participating nodes is large, and to reduce N linearly to one near the end of the cycle when no more transmission chances will be offered. Sample results of this strategy are shown in Table III-2. As an example, for the first group in test No. 3 with 20 starting points in each slot and groups of 4 slots, there are only 80 unique starting points. If the random number is selected from a range of 1 to 240 ($N = 3$), only 1/3 of the nodes will have transmission permission with the remainder waiting for the next group (where the selection range will be smaller). The results of Table III-2 show that it is possible to achieve a 7-percent near-packet loss rate with a transmission inhibition rate between 1 and 2 percent.

D. ANALYTICAL RESULTS RELATED TO THE RANDOM ACCESS PROTOCOL

In addition to simulation, analysis has been carried out to better understand the performance and trade-offs for the distributed random access protocol described in Sec. B. Results are presented and discussed in this section. Three topics are addressed: (1) The probability of unused slots within a single group; (2) success and collision statistics for a single slot; and (3) channel utilization by the time quanta introduced at the beginning of slots to control collisions. The analysis makes no distinction between near and far neighbors nor does it consider propagation delays.

Consider a single node with $n - 1$ one-hop neighbors and assume that a single transmission group consists of s slots. Let P be the probability that an individual node will attempt to use any of the slots. For the protocol described in Sec. B, nodes randomly select a number from one to $m*s*N$, where m is the number of time quanta at the start of a slot and N is a positive number, so that $P = 1/N$.

The relationship between the number of unused slots and the system parameters n, s, and P is easy to obtain. If $k < n + 1$ is the number of nodes which elect to schedule a transmission then k is a random variable with a binomial distribution:

$$b(k;\ n, P) = \binom{n}{k} P^k (1 - P)^{n-k} \quad . \qquad \text{(III-1)}$$

TABLE III-2

SIMULATION RESULTS WHEN THE PROBABILITY OF A NODE SCHEDULING A TRANSMISSION GOES LINEARLY FROM 1/N (N > 1) FOR THE FIRST GROUP TO ONE IN THE LAST GROUP IN A CYCLE. AVERAGE NUMBER OF NEAR NEIGHBORS = 5; AVERAGE NUMBER OF FAR NEIGHBORS = 13.

Test No.	m Starting Points in Slot	s Slots/ Group	N Ratio	Percent Far-Packet Collisions	Percent Near-Packet Collisions	Percent Inhibited Transmissions
1	20	8	4.0	19.7	18.6	<1.0
2	20	3	5.0	13.3	10.0	1.0
3	20	4	3.0	16.7	8.6	1.0
4	20	1	20.0	10.8	7.6	<1.0
5	20	2	10.0	11.4	7.8	<1.0
6	20	6	3.0	9.6	6.8	5.0
7	20	4	4.0	11.0	6.4	3.0
8	20	3	7.5	9.9	6.6	1.8

For a fixed k the probability that all slots are used is†

$$P_o(k, s) = \sum_{\nu=0}^{s} (-1)^{\nu} \binom{s}{\nu} \left(1 - \frac{\nu}{s}\right)^k \qquad \text{(III-2)}$$

and the probability that exactly e slots are empty is

$$p_e(k, s) = \binom{s}{e} \left(1 - \frac{e}{s}\right)^k p_o(k, s - e) \quad . \qquad \text{(III-3)}$$

Taking both the selection of nodes and the selection of slots by nodes into consideration, the overall probability of exactly e unused slots becomes

$$f_e(n, s, P) = \sum_{k=s-e}^{n} b(k;\ n, P)\ p_e(k, s) \qquad \text{(III-4)}$$

where the lower limit of the sum recognizes that if $k < s - e$ then more than e slots must be empty. The expected number of unused slots is then

$$E(e) = \sum_{e=1}^{s} e f_e(n, s, P) \quad . \qquad \text{(III-5)}$$

Figure III-2 shows E(e) normalized by s to represent the fraction of occupied slots. It is shown as a function of s/nP for each of three values of n. The case n = 20 is quite close to the $n = \infty$ curve. For values of $n < 5$, the curve continues to drop lower with the breakpoint moving to the right. We do not consider these smaller values of n any further in the present context.

It is desirable to have a small fraction of unused slots while simultaneously avoiding collisions. If s and n are fixed, P should be selected large enough to achieve the desired slot usage, but no larger since larger values result in more collisions. If the fraction of unused slots is to be on the order of 0.05 then, roughly, P = 2.5 s/n. If this gives $P > 1$, then 95-percent use cannot be achieved without using fewer slots or more nodes. Suppose that

† W. Feller, An Introduction to Probability Theory and Its Applications, Vol. I, 2nd Edition (Wiley, New York, 1957), p. 92.

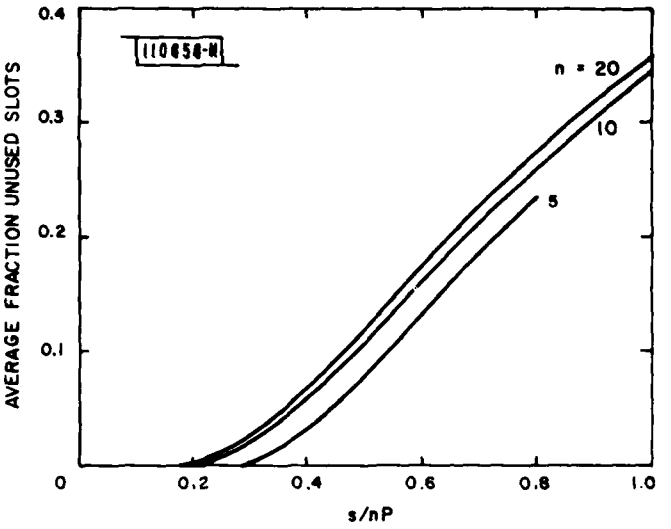


Fig. III-2. Average fraction of unused slots in a single broadcast cycle vs number of slots, s, normalized by the expected number of nodes, nP, choosing to attempt transmission in the cycle. Curves shown for number of nodes n = 5, 10, 20.

more unused slots can be tolerated, for example 35 percent unused. Then P = s/n. Although 35 percent unused may seem large, due to the need to avoid collisions it is more typical of good adjustment of system parameters than 5 percent unused would be. This point is addressed again in the following discussion.

The above analysis of slot utilization is not sensitive to the fact that the n nodes are embedded in a larger network and that the detailed actions of some if not all the n − 1 neighbors may be influenced by other nodes which are further removed in the network. That is not true of the analysis to follow. We arbitrarily consider the same n nodes but assume that they are completely connected to each other and disconnected from other nodes. The extra internal connectivity we assume for analysis should roughly compensate for ignoring links to nodes external to the set.

Consider a single slot within a group of s. Success and collision statistics have been analyzed for such a single slot by introducing one further approximation. The approximation is that a probability of P as discussed above

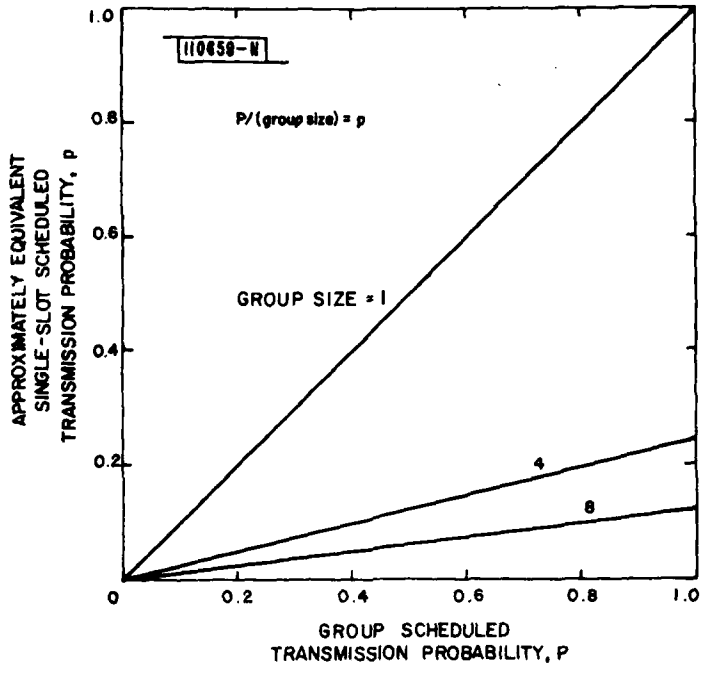


Fig. III-3. Single-slot scheduled transmission probability, p, which is approximately equivalent to a slot group transmission probability, P.

is equivalent to a probability p = P/s from the viewpoint of a single slot. This simple relationship is shown in Fig. III-3. It is approximate because slot occupancy for different slots in a group is not independent. For example, if P = 1 the total number of nodes attempting transmission in a group is n but if p = P/s is used independently on each slot the total number may be < n.

Given these approximations, expressions have been obtained for the probabilities of collisions (including zero collisions which is the same as successful transmission) for a single slot. Again let m be the number of possible start times within a slot. Let k be the number of nodes which attempt to use the slot. A c-fold collision will occur when the first start time with nonzero occupancy has been selected for use by c nodes. For c = 1 there is no collision. Given k and m, the probability of a c-fold collision can be shown to be

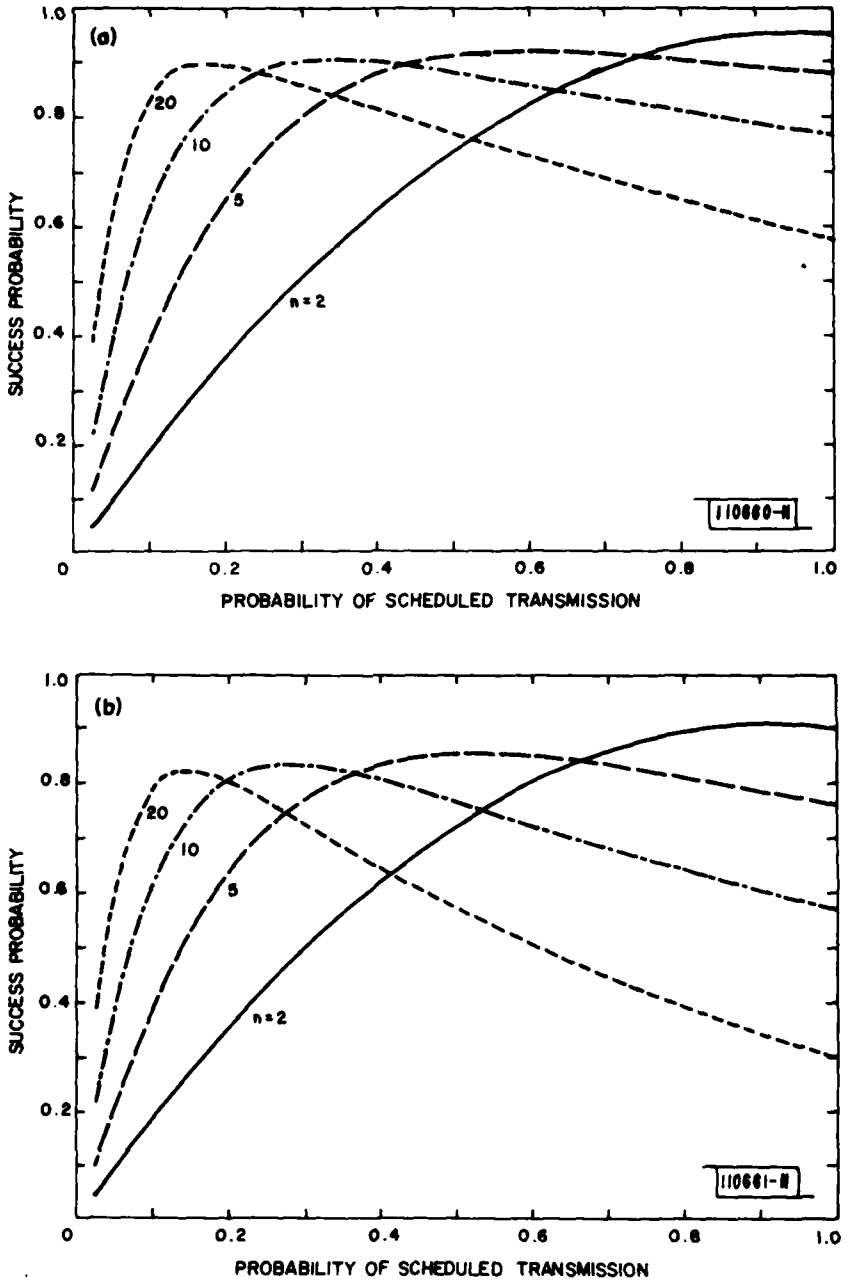


Fig. III-4. Probability of one node successfully transmitting in a single slot as a function of the probability of a scheduled single-slot transmission for individual nodes. Shown for n = 2, 5, 10, 20 participating nodes. (a) Twenty start times used for collision avoidance; (b) ten start times; (c) five start times.

$$g(c;\ m, k) = \sum_{v=1}^{m} \frac{1}{m^c}\binom{k}{c}\left(1 - \frac{v}{m}\right)^{k-c} \qquad \text{(III-6)}$$

with the convention that zero to the zero is unity and to any other power is zero. But the overall probability of a c-fold collision must also take into account the binomial distribution of k given p. Taking this into account gives the probability of a c-fold collision as

$$g_c(m, n, p) = \sum_{k=1}^{n} g(c;\ m, k)\ b(k;\ n, p) \qquad \text{(III-7)}$$

where b is the previously defined binomial distribution.

Figure III-4 shows plots of g_1, probability of successful transmission, as a function of p for several values of m and n. Several observations can be made about those curves.

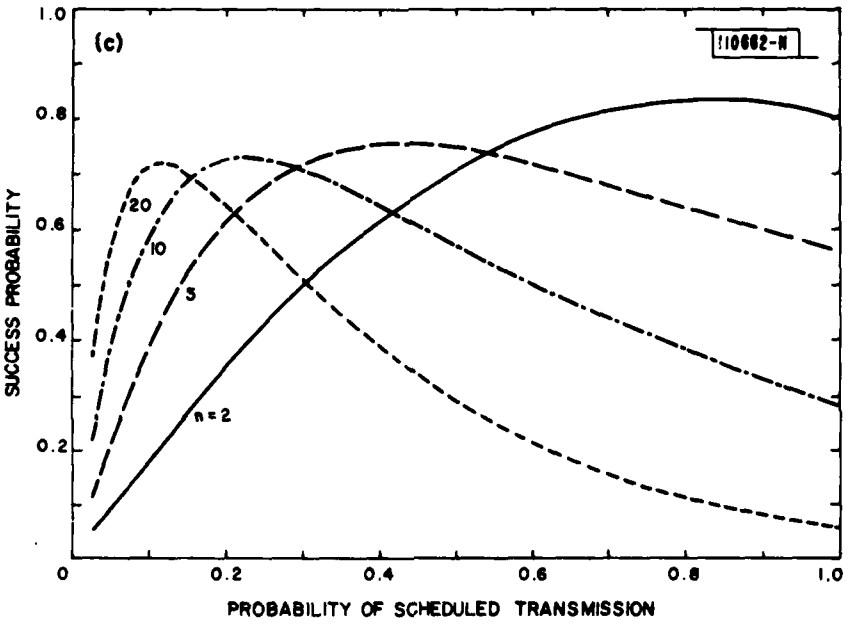


Fig. III-4. Continued.

There is a p which maximizes success and it decreases with the number of nodes. For p greater than the maximizing value the success rate drops due to collisions. For smaller p values the success rates drop due to failure to transmit. Maxima are broader for smaller numbers of nodes but, for the same value of m, the peak values, although larger for smaller numbers of nodes, are not very sensitive to the total number of nodes.

For two nodes and for p = 1 the success probability, $g_1(m, 2, 1)$, is $1 - 1/m$ which is roughly equal to the peak success values achieved by any number of sensors. This means that once m is on the order of 10 to 20, large increases in m will be needed for marginal improvements in success rates. But the use of many possible start times consumes some of the available channel, having much the same effect as a reduced transmission success rate in terms of throughput.

Figure III-5 shows the channel used by the m possible start times in a slot. For a 10-ms slot the curves shown correspond to 50-, 100-, 150-, and

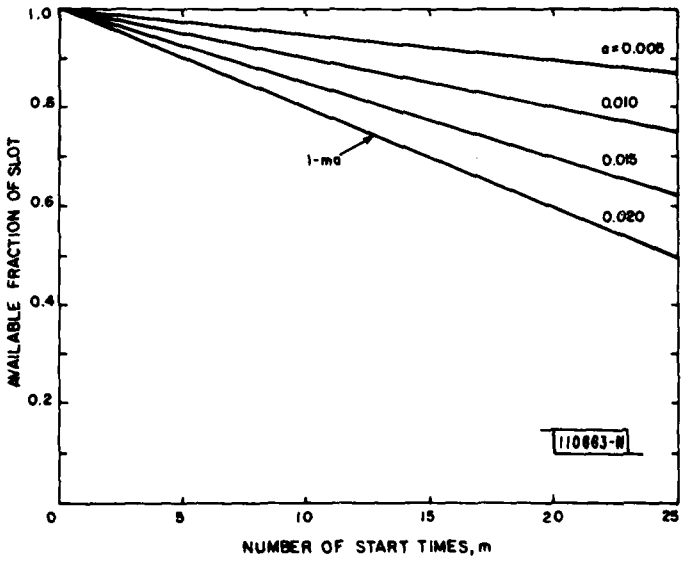


Fig. III-5. Fraction of each slot available for packets vs number of start times used per slot for collision avoidance. Parameter a is fraction of a slot corresponding to time between adjacent start times.

200-μs intervals between possible start times. Suppose 100-μs intervals are used and it is proposed to increase m above 20 to achieve a higher success rate. At m = 20, 20 percent of the channel is being used for collision control and the maximum achievable increase in success rate is on the order of 5 to 10 percent. To obtain only half of that by doubling m would consume another 20 percent of the channel and this would not be a profitable trade-off.

Transmission success is not the only parameter of interest. Collisions and the cost of collisions must also be considered. Suppose a collision of multiplicity c takes place. The most optimistic result would be that the c colliding packets would be lost but the remaining k-c nodes would detect at least one of the colliding packets and suppress their scheduled transmissions to await a later slot. The most pessimistic result would be that all k nodes transmit and collide. The expected number of packets lost due to collisions in these two extreme cases have been calculated and are shown in Figs. III-6 and -7. The formulae for these two cases are

$$E^{opt}\{c \mid c > 1\} = \sum_{c=2}^{n} c\, g_c(m, n, p) \tag{III-8}$$

and

$$E^{pes}\{c \mid c > 1\} = \sum_{k=2}^{n} k\, b(k;\, n, p) \sum_{c=2}^{k} g(c;\, m, k) \quad . \tag{III-9}$$

Figures III-6 and -7 show that the expected number of collisions can be sizeable for the probability values which maximize success rates. This is particularly so for the pessimistic case. For example, consider n = m = 20. As can be seen in Fig. III-4 the optimum p is about 0.17 with a corresponding success probability of about 0.89. The corresponding expected number of packets lost due to collisions is 0.36. These 0.36 packets and their information content are lost forever. The cost of a collision is considerable whereas a transmission which is not attempted will have another chance in the next slot.

One approach to selecting p might be to optimize a utility function such as

$$u = g_1(m, n, p) - w\, E^{pes}\{c \mid c > 1\} \tag{III-10}$$

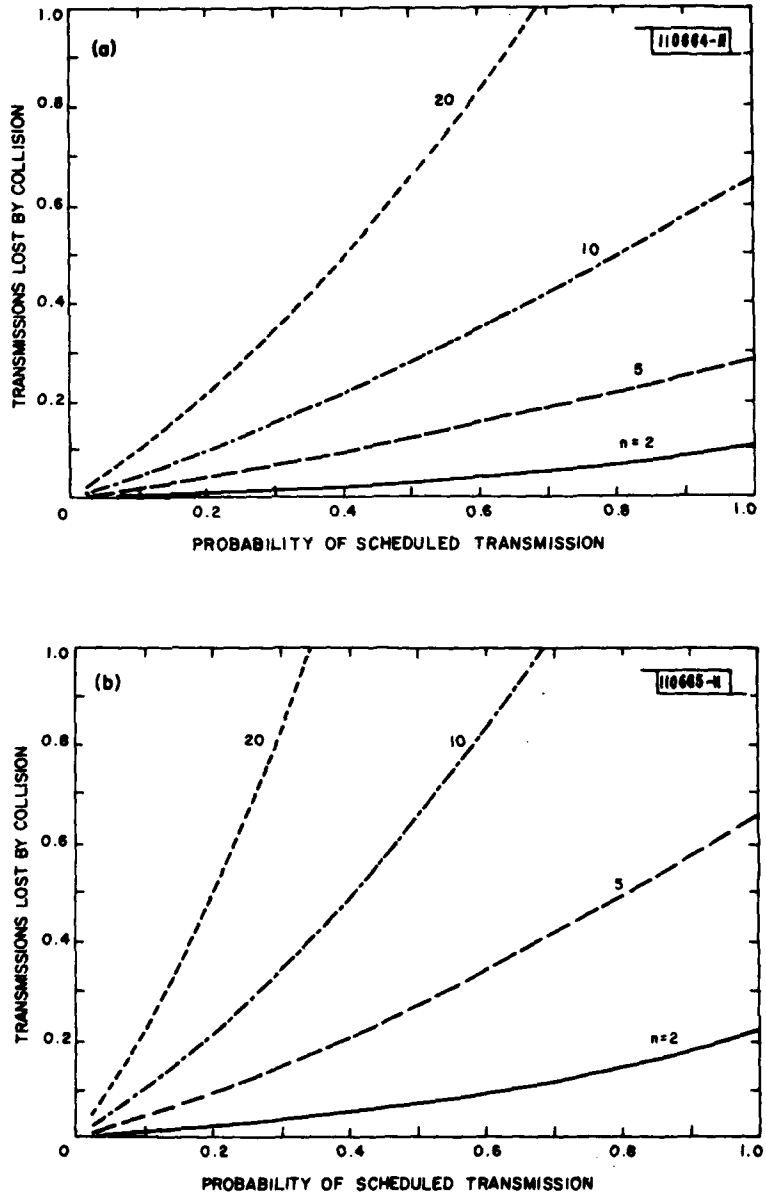


Fig. III-6. Expected number of transmissions lost by collision as a function of probability of a scheduled single-slot transmission for individual nodes (optimistic model). Shown for n = 2, 5, 10, 20 participating nodes. (a) Twenty start times used for collision avoidance; (b) ten start times; (c) five start times.

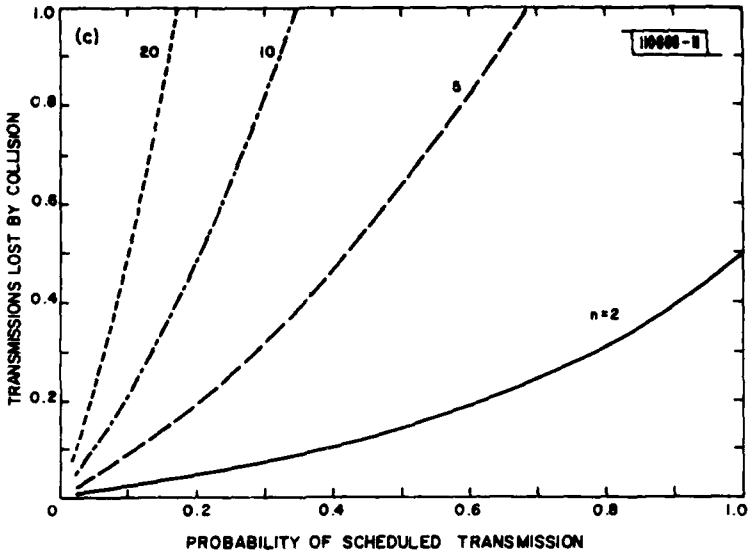


Fig. III-6. Continued.

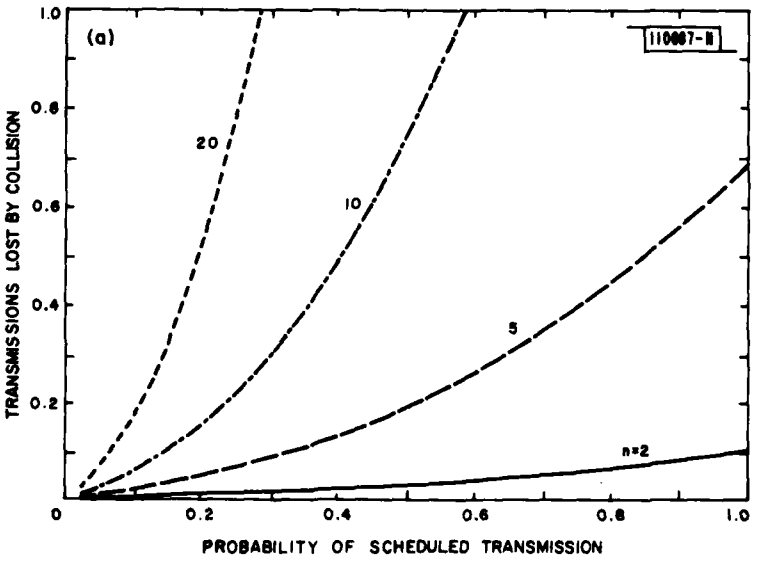
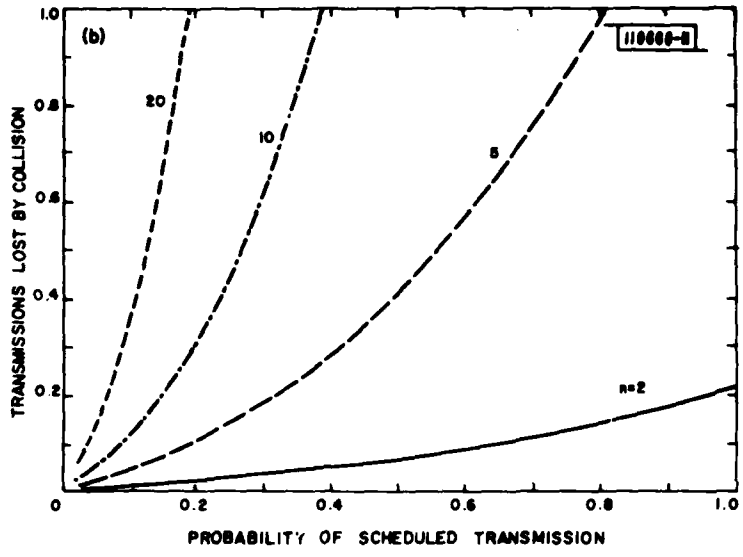


Fig. III-7. Expected number of transmissions lost by collision as a function of probability of a scheduled single-slot transmission for individual nodes (pessimistic model). Shown for n = 2, 5, 10, 20 participating nodes. (a) Twenty start times used for collision avoidance; (b) ten start times; (c) five start times.



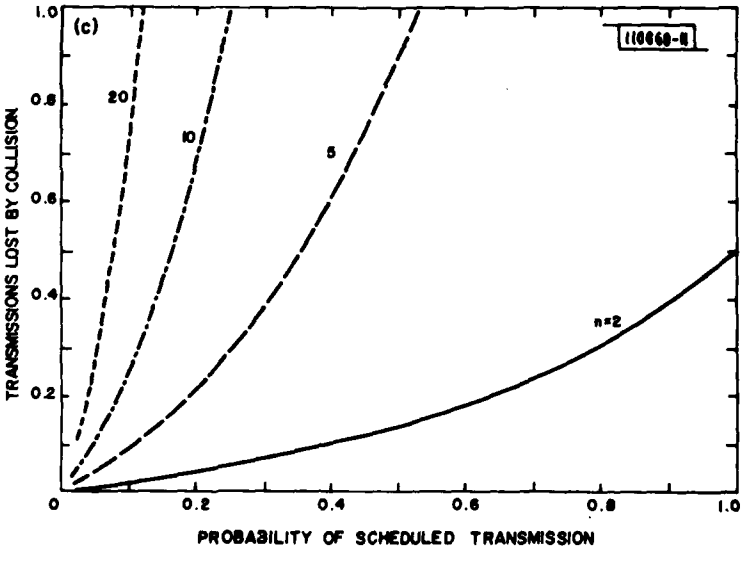


Fig. III-7. Continued.

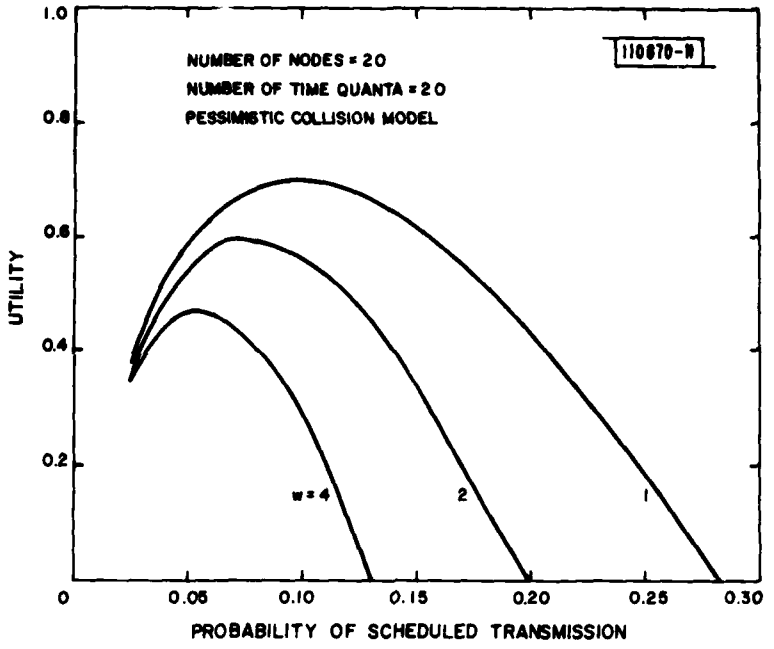


Fig. III-8. Utility (expected number of successful transmissions per slot minus expected number of transmissions lost due to collisions) as a function of probability of a scheduled single-slot transmission for single nodes.

which is the expected number of successful transmissions per slot penalized by w times the expected number of collisions per slot. This utility function is shown in Fig. III-8 for the case m = n = 20 and w = 1, 2, 4. For w = 1 the optimum choice of p in terms of u is reduced from 0.17 to 0.1. This corresponds to reducing collisions per slot by almost a factor of 3 to 0.13 at the cost of reducing success probability by only 0.06 to 0.83. For w = 4 the optimum p is further depressed to 0.05. The corresponding success rate and expected number of collisions are reduced to 0.63 and 0.04, respectively.

Now reconsider groups of slots and multiple groups in a cycle in terms of the above single-slot analysis. For p = 0.05 and s = 4 we get P = 0.2. If n = 20 we get s/nP = 1. As earlier discussed with respect to Fig. III-2, this corresponds to about 35-percent unused slots in a group. As just noted above it also corresponds to 63-percent success rate and 4-percent collisions. This

success rate can be substantially increased only at the cost of an unacceptable increase in collisions. A packet which collides is forever lost but, except for the last group of a cycle, a suppressed transmission will have another chance.

The overall broadcast scheme does not deal with fixed values of P and n through an entire cycle. The above discussions with n = 20 relate to the start of the cycle. As the cycle continues n is reduced by successful transmissions and collisions and P is gradually increased. The initial P is selected so that the number of nodes remaining at the last group is small and as many successful transmissions as possible are achieved in the cycle. Suppose there are n = 2 nodes left for the last group in the cycle. For P = 1, s = 4 this gives s/nP = 2, corresponding to considerable unused slots, very low collision probability, and a reasonably high probability that both packets will be sent in the last group of four slots. Specifically, in this case the single-slot success probability, using p = P/4 = 0.25, is 0.43 giving an average transmission of $0.43 \times 4 = 1.7$ packets in four slots and only an expected number of packets lost by collision of $0.006 \times 4 = 0.024$. We note that if success probability is kept in the 0.4 to 0.6 range throughout the cycle while keeping the collision rate small, then the number of packets handled successfully on a complete cycle will be about one-half of the complete number of slots in the cycle.

E. LARGER-AREA INFORMATION DISTRIBUTION AND COLLECTION

In the earlier description of channel utilization, 72 out of 100 time slots were assigned for sensor information transfer leaving 28 slots for regional area information distribution to and from an access node. The following outlines in a very specific context a scheme which we plan to generalize to provide such services in a general DSN.

Consider a quasi hexagonally packed grid of nodes with an access node at the center. In this case 3 successive rings around an access node will contain 6, 12, and 18 nodes for a total of 36 nodes to be contacted via regional broadcast. Of the 28 available time slots, take 10 for the one-to-many function. The first slot will be used to transmit to the first ring. But transmission

from all first ring transmitters in the second available slot could lead to serious contention problems. A way around this is to divide the approximately 6 nodes on the ring into 3 pairs where pairs are roughly diagonally opposite each other with respect to the access node. The next 3 slots can then be assigned to the pairs with little fear of contention at the second ring nodes. This leaves 6 time slots to get the message out to the third ring. Rather than continue to rely on site geometry to arrange a TDM partition one could at this juncture go to the type of random access schemes proposed for the sensor data exchange. This approach becomes more useful if the regional coverage requirements extend beyond 3 "rings."

As a message radiates out from the access node, each repeater can add to the header section the transmitter ID from which the broadcast was received. In this way each node obtains a route back to the access node. This route, acquired with low overhead, can be used to implement the report-back function as described below.

The remaining communication service is the report-back or collection function. Its purpose is to collect summarized and refined local area views from several tens of surrounding nodes. In the situation hypothesized here there are 18 time slots allocated to communicate back approximately 36 local views. Note that as the information funnels toward the access node, contention for the radio channel increases. Fortunately, packet collisions can be avoided by utilizing "receiver-directed" communications. During report-back, nodes will listen on their private codes but they will transmit using the listening code of the intended receiver. In this way communications intended for one receiver will not interfere with those of another receiver.

During the regional information distribution from the access node, the communication topology is a tree with the access node as the root. Moreover, each node knows the transmitter ID from which it received its copy of the broadcast as well as the number of hops to the access node. But for report-back there are multiple return branches leading to a node and, consequently, nodes trying to access this node can experience contention. Therefore, some further coordination is needed to implement the report-back function.

The following scheme appears to solve the report-back problem. Each node nominally has about 6 near neighbors which for present purposes is considered the maximum. The 18 remaining time slots can be divided into 3 groups of 6 slots each. For report-back, each node will independently assign one of 6 slots to each of its near neighbors. Thus there will be no possibility of contention because of the unique CDM/TDM assignment. In the first group of 6 report-back slots, those nodes that were reached via 3 hops (last link) will transmit in their unique CDM/TDM space. When this is completed the nodes in the second ring will combine the information from the third ring nodes with their own information and transmit "value-added" packets during the next 6 slots. In a similar manner the first ring of nodes will refine the data and provide transmission in the last 6 time slots. In the report-back mode, the successive refinement of relayed data by combining the received data with local data will help limit the total amount of data sent over the radio channel. The processing time to produce value-added packets may require distributing the different groups of report-back slots throughout the entire DSN cycle rather than clustering at the end as shown in Fig. III-1.

F. INVESTIGATION OF TDMA ORGANIZATION

Previous sections have considered a distributed approach to obtaining high channel utilization with a low collision rate in a broadcast network. Another channel access scheme which can achieve high usage and zero collisions is time-division multiple access (TDMA). In this technique each node is assigned transmission time from a set of time slots in such a way that no two nodes which are simultaneously transmitting will have a common one-hop neighbor.

A TDMA scheme was initially proposed as a candidate for the organization of the DSN communication channel.† A problem with that approach is how to assign the time slots to nodes in a general distributed network. But except for this difficulty TDMA remains a viable option, so we have further considered the assignment problem.

† Distributed Sensor Networks Semiannual Technical Summary, Lincoln Laboratory, M.I.T. (30 September 1978), DDC AD-A070455/1.

Let S, k = 1, . . . , K be a set of nonintersecting time intervals (slots). Let N be the number of nodes in the network. The problem is then to assign the K slots to the N nodes, reusing slots more than once, so that if node i broadcasts in its assigned slots it will be received by all its one-hop neighbors, with no interference from other broadcasts. Also, one would like to do this with the smallest possible value of K.

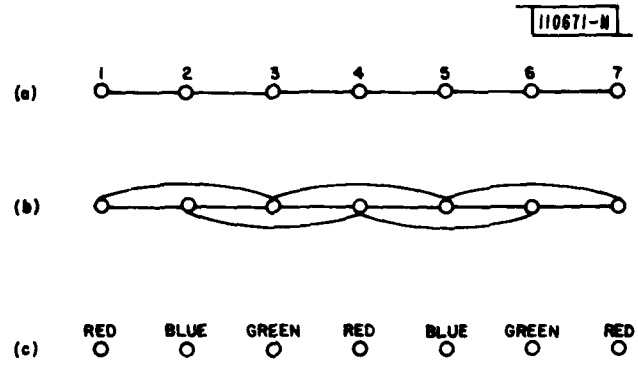


Fig. III-9. Noninterfering TDMA assignment in a broadcast network: (a) Example of one-hop network graph; (b) corresponding one- and two-hop network graph; (c) minimal noninterfering coloring of (a) which is also a minimal coloring of (b).

Figure III-9(a) shows a specific simple example of a graph with vertices representing nodes of a network and links representing one-hop neighbor connections. We call it a one-hop network graph. The two-hop network graph of Fig. III-9(b) is easily obtained by adding direct links between any pair of nodes in the one-hop graph which are connected by a two-link path. The TDMA organization problem is equivalent to vertex coloring problems for the two-hop network graph. The K slots can be assigned to nodes in a collision-free manner if and only if there is a K-coloring of the N nodes of the two-hop network graph, with no two adjacent nodes having the same color. In graph theory terms the smallest possible value of K is the chromic number of the

two-hop network graph. The chromic number of the two-hop graph of Fig. III-9(b) is 3 and a 3-coloring of that graph is shown in Fig. III-9(c). We call this a minimal coloring of the two-hop network graph.

Unfortunately, except for planar graphs and other specialized graphs, not enough is known about how to determine the chromic number of a graph and how to achieve the minimal coloring. One upper bound is supplied by Brooks' Theorem,† which states that any graph which is not complete or a circuit of odd length is R-colorable where R is the largest valency of any vertex in the graph. Valency is the number of links attached to the vertex. For the example of Fig. III-9 the valency of the two-hop graph is 4, meaning the graph is 4-colorable. But as is seen in the figure it is also 3-colorable.

We now describe an algorithm which is distributable and will always find a coloring for the two-hop network graph as long as the number of colors is one greater than the maximum valency of Brooks' Theorem. By distributable we mean that it appears that protocols can be developed for distributed, asynchronous implementation of the algorithm.

Let $i = 1, 2, \ldots, N$ identify the N nodes of a network and let $k = 1, 2, \ldots, K$ identify the K colors available for coloring the nodes of the two-hop network graph. Let L(i) be the list of triples

$$L(i) = \{[j, a(j), b(j)] \mid j = \text{one-hop neighbor of } i\}$$

where a(j) is a color and b(j) is a variable taking values "COLOR_ASSIGNED," "TRIAL_COLOR," and "CANT_ASSIGN." Let T(i) be the list of lists

$$T(i) = \{L(j, i) \mid j = \text{neighbor of } i\}$$

where L(j, i) is L(j) with the entry for i removed. Finally, let L(i; n), T(i; n) be the L(i) and T(i) at stage n of the algorithm to be described below.

For n = 0 each node randomly selects one of the K colors and sends [i, a(i), b(i)] to its neighbors. The b(i) are set to "TRIAL_COLOR" in all cases. Each node, i, can then create L(i; 0) and transmit it to all its one-hop neighbors. Each node then creates its own T(i; 0). If a(i) is not equal to any

† R. L. Brooks, "On Colouring the Nodes of a Network," Proc. Cambridge Phil. Soc. 37, 194-197 (1941).

of the colors in L(i; 0) or T(i; 0), then a(i) is unchanged and b(i) set to "COLOR_ASSIGNED." If a(i) is equal to the color of an entry of L(i; 0) or T(i; 0) and i is larger than the identifying index of the vertex with the matching color, then also a(i) is unchanged and b(i) is set to "COLOR_ASSIGNED." Otherwise b(i) is left "TRIAL_COLOR." When this is done for all vertices we go on to the next steps of the algorithm in which each node which has b = TRIAL_COLOR selects a new trial color from those available to it. Available colors are all colors except those in L(i; 0) or T(i; 0) with b = COLOR_ASSIGNED. Nodes with new trial colors send [i, a(i), b(i)] to neighbors. Modified L(i; 1) are then broadcast, updated T(i; 1) calculated, and the algorithm continues as in the previous cycle.

Subsequent cycles are similar. If there are no colors available for a vertex it takes b = CANT_ASSIGN and no longer participates in the algorithm. The algorithm terminates when all nodes have b = COLOR_ASSIGNED or CANT_ASSIGN.

This algorithm has the following two elementary properties. First if R is the maximum valency of any node in the two-hop network graph then the algorithm will successfully color the graph if R + 1 colors are supplied. Second, the algorithm will terminate after at most N cycles. The first assertion follows from the fact that no matter what R colors are assigned to the R neighbors of a vertex, there is a (R + 1)th color available. The second follows because at each interaction at least the vertex with the highest index and b = TRIAL_COLOR will take one b = COLOR_ASSIGNED or b = CANT_ASSIGN.

The above algorithm has been implemented and tested. For convenience, a linear network of nodes such as that of Fig. III-9 was used but there are no fundamental differences for a two-dimensional network. A network of 1000 nodes was considered. The maximum number of one-hop neighbors, M, was one adjustable parameter and the one-hop neighbors were taken to be the closest M nodes. Unique identification numbers were randomly assigned to all nodes using a pseudorandom number generator. Runs were made with 10 colors (slots) and with 20 colors.

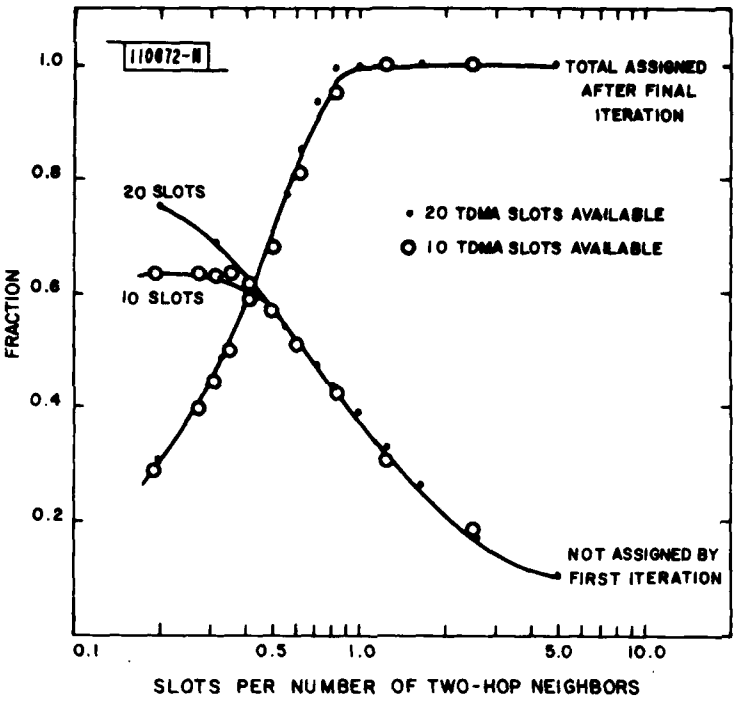


Fig. III-10. Simulation results for distributable TDMA slot assignment algorithm. Results for linear array of 1000 nodes. Worst-case convergence in five cycles.

The results of the computer runs are summarized in Fig. III-10. It shows two basic curves. One curve is the fraction of nodes with assigned slots at the termination of the algorithm, which actually required only five cycles for the worst case considered. The other is the fraction which is neither "COLOR_ASSIGNED" nor "CANT_ASSIGN" at the end of the first cycle, indicating that a high fraction of the assignments ever made are made on the first iteration. The curves are shown as a function of S/R where S is the number of slots and R, as defined in the previous discussion, is the largest number of two-hop neighbors for any node in the network. In the linear network case R is about 2M. For a two-dimensional network it would be on the order of 4M. For S/R > 1 all assignments are made as expected. As S/R is reduced below unity the algorithm begins to be unable to assign slots to all nodes.

Unfortunately, the fraction of assigned slots drops rather quickly even when S/R is in a region for which we know perfect assignments are possible. For the linear network of the simulation, a perfect assignment is possible as long as the number of slots is at least as large as M + 1, or roughly as long as S/R is greater than 0.5. For S/R = 0.5 the algorithm achieves only 60 percent assignment, although one can think of schemes for sharing slots and even broadcasting with some conflicts which could result in a viable system built up from the basic assignment algorithm.

The primary reason that only M + 1 slots are actually needed is that single-hop connectivity is localized and somewhat regular. If the M connections from a node went to M randomly distributed other nodes the number of slots needed would typically increase toward R, the Brooks bound. With regular, clustered, one-hop connectivity one need only cyclically assign slots $1, 2, \ldots, M, M+1, 1, 2,$ etc. to the linear network. A little experimentation shows that replacing some local connections with more remote connections increases the required number of slots. The fact is that our algorithm will work very well for networks with random single-hop connectivity, which require a number of slots approaching the Brooks bound. It remains to be determined if some way can be found to exploit the more clustered connectivity of a DSN to obtain near optimal performance for that case.

IV. NODAL PROCESSORS AND DISTRIBUTED SOFTWARE

A. ADVANCED NODE ARCHITECTURE

We are investigating options for an advanced DSN nodal architecture which is flexible, provides for considerable nodal processing power for reasonable design and construction cost, accommodates growth requirements, adapts to a range of requirements, and which serves as a model for future low-cost and low-power DSN nodes. We are considering a modular multiple-processor approach in which increments in processing power are achieved by adding processors and distributing tasks and algorithms among them. The plan is to design and develop a prototype system which can be integrated into the DSN test bed. We are intending to use Motorola MC68000 processors to implement the elemental processors. The prototype system will allow us to develop, demonstrate, and test design features that will aid in maintenance of multiprocessor systems and features that will ease the task of developing reliable distributed software.

The interprocessor communication subsystem and the amount of private memory for each processor are important aspects of a multiple-processor system. In the following we examine candidate interconnection techniques and the question of memory size. The relative merits are presented for developing a DSN multiple-processor system based upon commercially available boards vs designing new boards. The Motorola Versamodules are the commercial boards used for the comparison.

1. Shared Memory Architectures

The existing test-bed nodes (Fig. IV-1) are built around a standard minicomputer architecture. The PDP-11/34 processor, memory, and I/O devices are connected to a bus (the UNIBUS) which contains address, data, and control lines. Extra processors (Motorola Versamodules) are connected to the bus through DMA interfaces. The PDP-11/34 is normally the bus master and may transfer ownership to DMA devices. The PDP-11/34 processor must initialize the interfaces and the PDP-11/34 memory is used for communication between all processors.

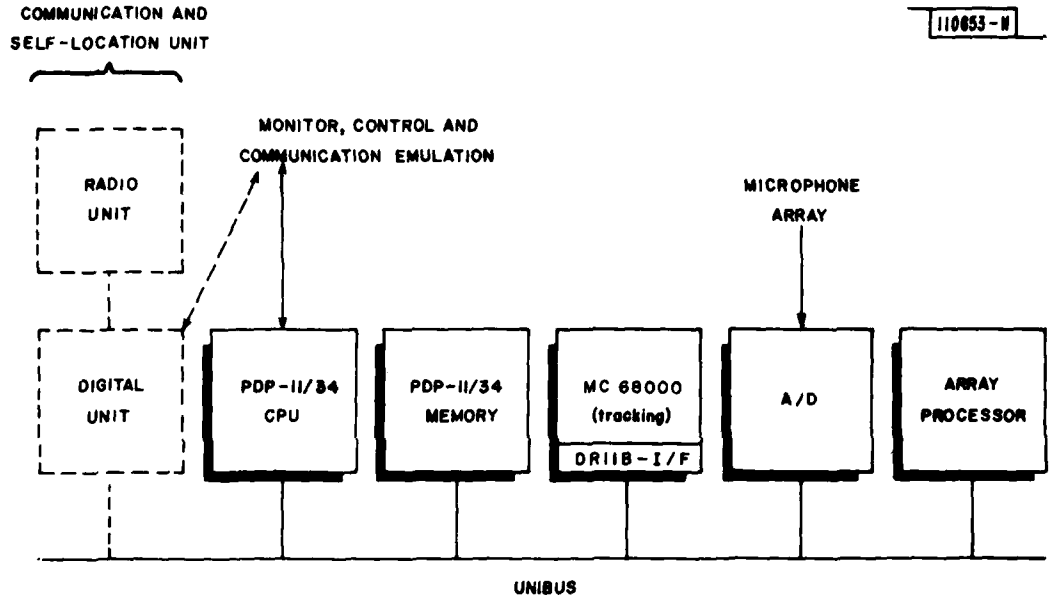


Fig. IV-1. Standard multiprocessor minicomputer architecture of existing test-bed nodes. Dotted items yet to be added. All data and processing traffic share a single bus.

One straightforward extension of the existing test-bed architecture could be considered as a candidate for future nodes. Most microprocessor manufacturers provide the capability of connecting multiple processors with a common memory (Fig. IV-2). As is the case in the current test-bed implementation, shared memory is used to communicate between processes, and as long as the shared memory occupies the same virtual address space in each of the processors, communication can be performed using global objects.

This approach conforms to the capabilities of standard Versamodules. Each Versamodule processor board is organized in a standard minicomputer architecture with local RAM and I/O devices. Initial versions of the processor board will have 32 kB of on-board RAM and 32 kB of on-board ROM. The RAM may be increased to 64 kB or even 128 kB although no schedule has been announced for such upgrades. In the following, for discussion purposes, we

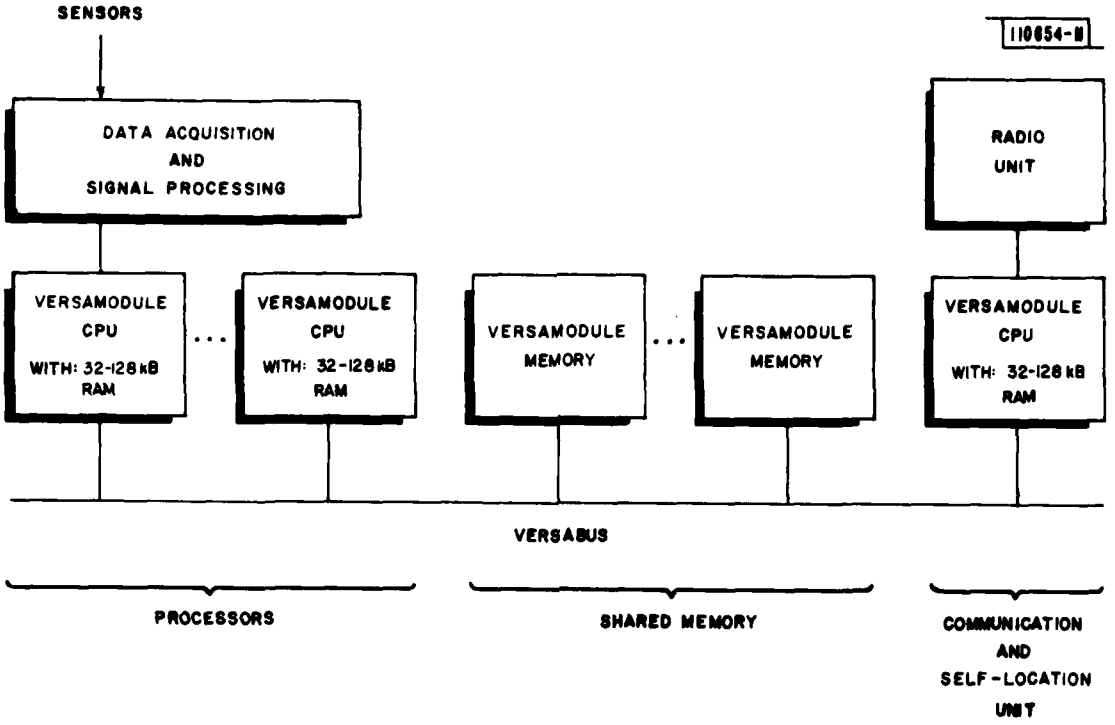


Fig. IV-2. Possible future node architecture based upon standard minicomputer architecture. Figure indicates possible realization based upon Motorola Versabus and Versamodules.

have generally assumed the availability of a processor board with 128 kB of on-board RAM.

A separate RAM memory board has been announced for use in conjunction with the initial (and subsequent) versions of the processor board. It can be used as an extension of the on-board private memory and as a memory shared between processors. Each Versamodule includes an interface to a Versabus that provides access to the separate memory boards. To communicate data from one Versamodule to another the data source must copy packets to the shared memory on the Versabus. The data destination must then

copy the packets from the shared memory to its local memory. When the Versamodule accesses a memory location outside its private memory, it must gain control of the Versabus in order to access the shared memory. The bus arbitration hardware requires that each Versamodule be assigned a fixed priority so that priority must be assigned on a processor basis rather than on a data-packet basis.

Copying operations between local memory and shared memory are limited to one-half the Versamodule read or write cycle time (half the accesses will be to local memory, half to shared memory). This limits processor to shared memory transfers to about 1.5 MB/s. But a packet transfer between processors requires two copy operations, one from the source to the shared memory and a second from the shared memory to the destination. This reduces the effective bus bandwidth to the fairly low rate of 750 kB/s.

2. Packet-Bus Architectures

One problem with the system described above is that 128 kB is not enough memory for a single processor module. An MC68000 processor is fast enough to support 256 kB or more of memory, depending upon the application. Also, based upon experience with the DAK2 system now operating in test-bed nodes and upon our plans for more advanced network software as discussed in Sec. B below, 32 to 64 kB of memory will be required for operating system and run-time support software. In this case a 256-kB module might replace 2 to 3 Versamodules with only 128 kB. A further advantage of having a large local memory would be realized through reduced software costs, which are most easily obtained by providing a larger, more capable run-time library and reducing the use of interprocessor communication.

The Versamodule local memory is not extendible by any means except the Versabus. Also, it is not possible to attach several Versamodule processors to the same Versabus and to use the Versabus to extend the local memories as well as to provide access to common memory. The reason is that the MC68000 design will cause the bus to become saturated when more than two processors are using it to access local memories. Therefore, the only way to increase the private memory on any Versamodule is to use the Versabus as a local

minicomputer bus and to find other means for interconnecting the processors. The interconnection between the processors need not be as general as a standard memory bus and can be simplified to a pure packet bus (Fig. IV-3). The standard I/O devices on a Versamodule are not capable of supporting a packet bus, so a custom-designed interface is required. Since packet copying can be done by DMA directly from one processor to another with no intervening shared memory the transfer rate can easily be 4 MB/s or more.

If the interprocessor communication must be handled with a specially designed interface, the attractive feature of using only commercially available hardware must be abandoned. In such a situation, we must evaluate the gains that might be achieved if the entire hardware system were redesigned. For example, if the processor card is redesigned, then a DMA interface to the packet bus and 256 kB of memory can be included on the board. Also, there are useful features such as a process timer which stops during interrupts and a memory protection system which allows faults in one process to be isolated from another process, which are unavailable in current Versamodules. In addition, many robustness and automatic testing features should be included in a system which is intended to eventually operate unattended in remote locations.

A rough estimate of IC counts bears upon the selection of a development approach. A completely redesigned processor board might consist of approximately 120 ICs: perhaps 60 ICs in the microcomputer, and 60 ICs in the packet-bus interface, I/O interface, and a few other features like memory protect and a process timer. The packet-bus interface module for the commercial hardware might itself consist of 100 ICs: 60 ICs in the packet-bus interface, I/O interface, and other features and 40 ICs in the Versabus interface. Thus, with the commercial hardware we have the disadvantage of a major subsystem being unobtainable from commercial vendors. More important, the cost, size, and power for complete systems would be larger than those obtained when the processor is redesigned. One might hope for reductions in cost, size, and power consumption by factors of 2 each with a unified, redesigned processor card simply due to the elimination of unnecessary and duplicated hardware.

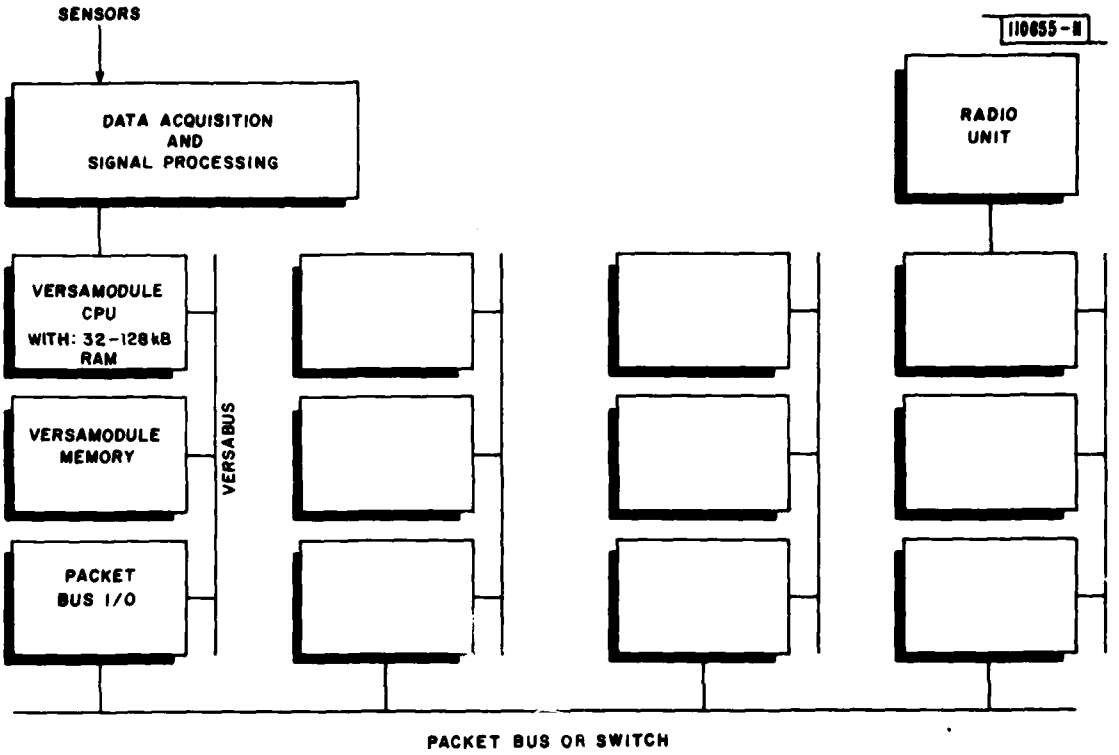


Fig. IV-3. Possible future node architecture with packet bus used for direct interprocessor communication. Figure indicates possible realization using Motorola Versabus and Versamodules.

3. Packet-Bus Design Details

The simplest packet-bus connection is a DMA connection between the memories of the transmitting and receiving processors. In this case the packet bus preempts the processors and forces them to relinquish the bulk of their memory during the packet transfer. Since the memories are dedicated to the transfer, the bus can be driven at approximately the cycle time of memories. If I/O operations are to be supported during packet transfers, either a separate I/O processor must be included or the processor must be able to sever its connection to the RAM participating in the packet transfer and have a small amount of separate RAM (perhaps 4 kB) for stack use and I/O data buffering.

The packet bus may be scheduled using a simple, low-bandwidth distributed arbitration scheme which may be overlapped with packet-bus transfers. The algorithm is simple and slow enough that a single-chip microprocessor (e.g., Motorola's 68120) may be able to perform the scheduling function and relieve the main processor of the burden of negotiating on every packet. Scheduling the packet bus resolves contention and, since the memories are dedicated to packet communication, there is no congestion problem.

A modification to this technique, which we have examined but do not recommend, removes the requirement that the memories of the participating processors be dedicated to packet communication. A buffer would be inserted between the processor's memory and the packet bus. A separate buffer area would be required for each priority level to avoid the problem that a low-priority message may fill the buffer, thereby blocking a higher-priority message. Each buffer must be large enough to store all bus transactions from the time the main memory becomes unavailable until the receiver can force the transmitter to stop. This may take several milliseconds. If we assume a 4-MB/s packet-bus transfer rate, a size of perhaps 32 kB is implied for each buffer. Even with only 4 priority levels, the total required buffer size is on the order of 128 kB. An intelligent controller is also implied in order to handle the protocol required to turn off the transmitter. Such an interface is comparable in size to the processor which is to be serviced, and the cost seems prohibitively large for the benefit derived.

4. Summary

There are substantial risks involved in using the commercial board level hardware as the basis for an advanced node configuration. The Versamodule local memory limitations, especially if only 32- or 64-kB RAM versions are available, would place unacceptable constraints upon the run-time library and the size of application software modules. The need to design a custom packet-bus interface would reduce the advantages of using standard commercial boards. A large number of Versamodules could become a reliability problem. The use of commercial processors would make it difficult or impossible to develop and

evaluate a number of advanced features relating to reliability. A node based upon the commercial hardware would involve a number of important compromises and would not serve as well as a model for future high-performance, low-cost, low-power nodes.

In light of these considerations, the currently favored approach is to develop a multiprocessor system using custom-designed processor boards interconnected with a bus providing direct data transfers between processor memories. There are other methods of interconnecting multiple processors, but we are unaware of any which provide a benefit sufficient to justify the increase in complexity with the concomitant increased cost of designing and maintaining both hardware and software. The system is highly modular and, initially, it can be constructed using only one module type. This module would contain an MC68000 processor, local memory, a packet-bus interface, and some general-purpose I/O. A memory expansion module can be designed and constructed when required.

B. DISTRIBUTED REAL-TIME SOFTWARE

The Real-Time Network Kernel (RENE)† is a distributed, real-time operating system including processor scheduling and memory management mechanisms and a higher-level interprocess communication protocol. The interprocess communication facility of RENE must serve the needs of higher-level operating systems and of application systems, in an environment where the communication medium may be anything from radio to shared memory. Development of RENE is evolutionary. The DAK2 software currently supporting application software in the test bed is a first step toward implementing and testing basic ideas concerning interprocess communication. The plan is for this to evolve into a system optimized for use in a distributed environment and implemented for both PDP-11's and MC68000 systems in the test bed. During the last six months the RENE design has been refined and coding has been completed for basic routines such as the processor scheduler and memory

† Distributed Sensor Networks Semiannual Technical Summary, Lincoln Laboratory, M.I.T. (30 September 1980), DTIC AD-A103045.

map mechanisms. See Sec. C for a report of related development effort on general-purpose support routines for DAK and RENE.

System reliability and resource management have been major concerns of the RENE design effort. Our approach to system reliability is that user processes recover from failures ("crashes") of server processes by reinitializing the server at a checkpoint and resuming operation as if the crash had never happened. The goal is to automate this process as much as possible so that application programmers need not expend great effort on the system reliability issue. The principal reason for this approach is the great difficulty of actually eliminating random crashes from a distributed system, particularly in an environment with changing hardware and software.

The RENE interprocess communication system is based upon queueable objects, which are structures that may be sent as a request from a user process to a server process, and returned as a reply from the server process back to the user process. When sent they appear on a queue in the server process address space, and simultaneously on another surrogate queue in the user process address space, with these two queues being logically connected to one another. While the server has the object there are essentially two copies in existence: one in the server address space and one in the user space. With a few variations introduced to respond to special resource management and reliability needs, we anticipate that queueable objects will form the basis for most distributed interprocessor communication, just as the procedure call and global variable currently form the basis for intermodule communication in virtually all nondistributed programs.

One important special class of queueable object is the parameter object. This object holds server status information that is set by the user process and must be reset if the server is reinitialized. A parameter object is sent from user to server, and held indefinitely by the server so its parameters may be used in processing subsequent request objects. If the user and server processes are in different virtual address spaces, the user's copy of a parameter object remains on the surrogate queue in the user space that is connected to the server queue in the server space. If the server crashes, retransmission of the parameter object can be automatic when the server is reinitialized.

Two options are now being considered as a means for changing parameter objects. One approach is for the user to send a second parameter object which is a copy of the first with appropriate modifications. The other approach is for the user to send a request object to the server to make modifications in a parameter object.

Another important special class of queueable object is the "connectable" object. This is like a request object, which is sent to a server and returned by the server. However, even after return the object remains connected to the server in the sense that the server may keep a copy of the object. The user and server copies of the object are said to be connected. Connectable objects obey a basic open-connect-disconnect-close protocol. A connectable object can also have its connection broken by the user, after which the object may be passed to another virtual address space and the connection reestablished. Thus, the final disconnect-close of the basic protocol may be preceded by a series of disconnect-reconnect actions. There are system functions for opening, connecting, disconnecting, reconnecting, and closing connection objects. Connectable objects are not kept on surrogate queues and can be freely passed to different processes. Surrogate queues themselves are a prime example of connectable objects. They contain a pathname string, and are connected by being sent to a directory server, which locates or creates the server to which the queue is finally connected. Connectable objects seem to be necessary for use as a resource allocation mechanism, as well as a foundation for surrogate queue objects.

The RENE design of processor and memory space managers will make use of parameter and connectable objects. Design work on these managers is now proceeding. The frequent state change involved with memory page swapping greatly stresses the ability of a design to make changes in server state. These particular managers are excellent test cases for any protocol purporting to facilitate distributed resource management.

C. OBJECT STRUCTURED DISCIPLINE DEVELOPMENT

The Object Structured Discipline (OSD) (SATS, 30 September 1980) is a set of run-time library functions and program development tools which now

support our Data Acquisition Kernel (DAK2), and will also support the Real-Time Network Kernel (RENE) which will eventually replace DAK2. At the present time OSD is a PDP-11 software system which is written largely in C. It will ev~ ually be converted for MC68000 use.

During the last six months we have refined the definition of a ty_typed object (see below) and the basic compilation tools, have implemented 64-bit integer arithmetic, error message objects, and some forms of character string objects, and have written additional documentation for the coding conventions of OSD and ty_typed objects.

A ty_typed object is a structure whose value contains type information about the object. We have rearranged this type information to allow much faster run-time integrity checks to be made to ensure that a pointer is indeed pointing at the right kind of object. In the case where one object is several types at once the previous form of integrity check took a relatively long time to test whether the object was of the proper type because only one type constant was actually stored in the object. The new object format reduces all integrity checks to a single comparison. Such integrity checks have been very useful to compensate for weaknesses in the strong type checker for the C programming language, to detect software errors that overwrite portions of memory, and in one important case to detect a hardware error.

We continue to improve our system for compiling binary files for different computers and operating systems from a single set of source files. To this end we have interfaced the standard C macroprocessor as a front end to the assembler so that our assembly language code can use the same conditionalization mechanism as compiler language code to adapt to different environments. We are also taking the "make" description files, which specify how to make binary files from source files, and embedding much of their contents in common files that can be included by different subsystems. To this end we have redesigned the feature which we added to the description language that allows one description file to include another. Lastly, we have worked on automating the processes of producing verifier listings, program code line counts, and program code listings.

In implementing 64-bit integer arithmetic for our 16-bit PDP-11/70 and PDP-11/34 computers, two simple lessons have been learned. The first is that functions such as those which add two operands and return the sum, although elegant, are inefficient, and should be replaced by functions which add one operand directly to a variable. The second lesson is that using short integer division hardware to implement fast long divisions is very expensive in terms of memory space. A 64- by 32-bit division function was 500 bytes longer when coded with a fast algorithm using 32- by 16-bit hardware divide instructions than when it was coded with a slow algorithm using only add and shift instructions.

The error message object we have implemented holds an error message or is in no-error state. The error message is in the form of an argument list to a formatted print routine: e.g., the call—

```
pc_error ("fi_open: file %s does not exist", pathname)  ;
```

specifies an error message consisting of a format character string in quotes and one argument, which is also a character string that will be substituted for "%s" when the message is printed. The error object actually stores the format string and following arguments separately. This reduces the amount of computation involved in creating an error message to a minimum, and makes it easy to extract the separate components. The format string specifies the type of error, so that other programs with error type specific recovery capabilities can function. Arguments are limited to numbers and character strings.

An error object has been included in each process object as the basis of an error control mechanism. A process is said to be in error state when its error object contains an error message. Each function is responsible for entering error state when it detects an error it cannot otherwise handle, which the function does by means of a call to pc_error as in the above example. Whenever a function enters error state, it should terminate current activities in an orderly fashion and make it appear that the function has done nothing but enter the error state. The function then returns, preferably with a value indicating an exceptional condition. Functions which discover that a function they have

called has entered error state should either correct the error and clear the error state, or behave as if they had entered error state. This clearing and error reporting system compensates for the lack of an exception-handling mechanism in the C language. It also has some advantages over typical exception-handling schemes, in that each exception comes with a message that is both printable and contains arguments for use in recovery. Simple character string objects have been implemented to support the error message objects.

V. MISCELLANEOUS

A. SENSOR COVERAGE REDUNDANCY

An earlier SATS† reported on DSN effectiveness as nodes become inoperative. Both adaptive communications and sensor coverage were considered and were examined by means of Monte Carlo simulations. It turns out that there is a simple and accurate approximation for the fraction of surveillance area which lies within a sensor range, and the analysis is easily extended to compute the fraction of the surveillance area within range of at least k sensors. The k = 2 case is clearly of interest for acoustic sensors which require two sensors to provide a location and initialize a track.

Assume that sensors each have an area of coverage, A, and are randomly distributed on the surveillance area, with a density λ, according to a two-dimensional Poisson process. The probability that a given point is within range of k sensors can be shown to be

$$1 - e^{-\lambda A}\left[1 + \lambda A + \ldots + \frac{(\lambda A)^{k-1}}{(k-1)!}\right] \quad . \tag{V-1}$$

This also can be interpreted as the fraction of the surveillance area which has coverage by k sensors. If λ_o is sensor density when all nodes are functioning and η is the fraction of functional nodes, then using $\lambda = \eta\lambda_o$ in the above expression gives the fraction of area with k-fold coverage as a function of η.

If nodes are on a hexagonal grid with spacing d and the sensor coverage is circular with radius r, then

$$\lambda_o = (\tfrac{4}{3})^{1/2}/d^2 \quad , \tag{V-2}$$

$$\lambda = \eta\lambda_o = \eta(\tfrac{4}{3})^{1/2}/d^2 \quad , \tag{V-3}$$

and

$$A = \pi r^2 \quad . \tag{V-4}$$

† Distributed Sensor Networks Semiannual Technical Summary, Lincoln Laboratory, M.I.T. (31 March 1979), DDC AD-A082395/5.

From this the fraction of area covered by at least one sensor is approximately

$$1 - \exp[-3.63\ \eta(\frac{r}{d})^2] \quad , \qquad \text{(V-5)}$$

where 3.63 has been used for $\pi(4/3)^{1/2}$. The fractional coverage for k sensors is

$$1 - \exp[-3.63\ \eta(\frac{r}{d})^2] \left\{ 1 + 3.63\ \eta(\frac{r}{d})^2 + \ldots + \frac{[3.63\ \eta(\frac{r}{d})^2]^{k-1}}{(k-1)!} \right\} \quad . \qquad \text{(V-6)}$$

For this approximation, using expressions which are valid for random distributions for the case of a regular hexagonal distribution, the fractional coverage is shown in Fig. V-1 for k = 1, 2 and r/d = 1.4. For the case k = 1, the analytical curve is higher than the curve previously obtained by simulation.

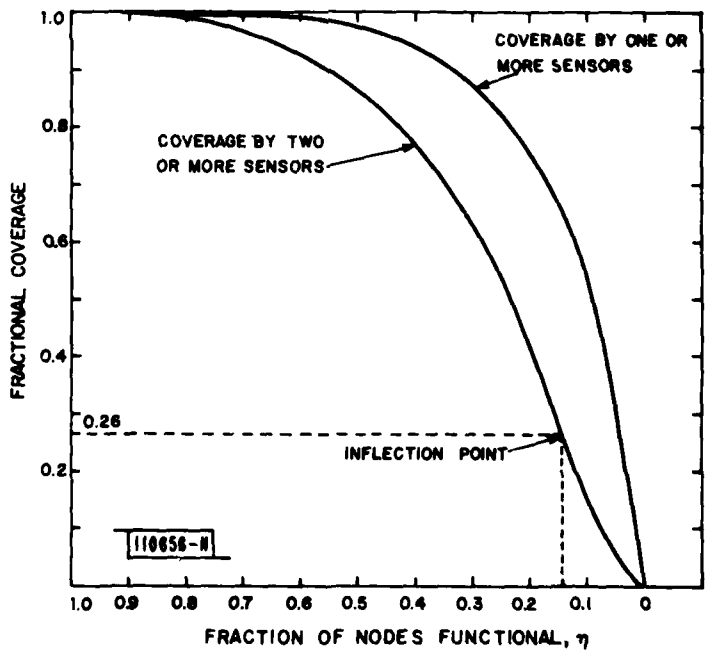


Fig. V-1. Analytical model of surveillance area coverage as a function of fraction of DSN nodes which are functional. Curves shown for r/d = 1.4 where r is sensor range and d is sensor separation on a hexagonal grid.

The reason is primarily edge effects in the simulation where coverage is not as complete at the edges. The earlier simulations did not consider k = 2 or more. For k > 1, the curve is more s-shaped with slope zero at $\eta = 0$ and a non-positive slope at $\eta = 1.0$. The curves for k > 1 all have inflection points with that for k = 2 at

$$\eta = \frac{1}{\lambda_o A} = \frac{1}{3.63 \left(\frac{r}{d}\right)^2} \quad . \tag{V-7}$$

The corresponding coverage by two or more sensors is 0.26.

B. EXTERNAL INTERACTIONS

The DARPA Packet Radio Project represents the current state of the art of distributed computer networks using broadcast communication for interconnection. Therefore we continue to follow this work closely. To this end we attended a three-day Packet Radio Working Group meeting to keep informed in that area.

We continue to interact with M.I.T. campus. We have participated in and given presentations at a Passive Surveillance Seminar series involving M.I.T. faculty and students as well as participants from industrial firms. An investigation of two-dimensional Maximum Entropy array processing is being carried out jointly with Professor Lim and one of his students and single and multisite tracking are being further investigated by Professor Tenney and his students. Also within the academic community, we have participated in and given presentations at an all-day meeting held jointly with the Distributed Decision Making Group at the University of Massachusetts which is under the direction of Professor V. Lessor.

The use of acoustic and seismic mechanical waves for passive battlefield surveillance continues to be of interest to many different groups and a NATO study group directed toward this area (RSG-11) has been formed. Under the auspices of RSG-11, acoustic and seismic experiments have already been done in Europe by participating countries and more will be done in the future. We have participated in these experiments by providing guidance and we have now obtained some of their data from two very small (0.5-m aperture) acoustic arrays and plan to analyze it to further evaluate acoustic sensor potential under

different environments and to investigate opportunities to exploit modern signal-processing techniques.

C. PDP-11/70 FACILITY

A PDP-11/70 running the Version 7 UNIX operating system continues to be our primary software development and analysis tool. The machine is heavily used and required several hardware additions during this reporting period.

A large number of processes are routinely run and more memory was required to alleviate poor response due to swapping. The system has now been upgraded to 1 MB and the situation has improved. Software development, and to a greater extent data analysis activities, necessitated the addition of a second tape drive and 300-MB disk system. An additional terminal multiplexer and short-haul modems have been procured and installed to provide terminal level service for user terminals and outboard computers such as the PDP-11/34's and MC68000's which are located in the test-bed nodes.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-81-210	2. GOVT ACCESSION NO. AD-A108 275	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Distributed Sensor Networks		5. TYPE OF REPORT & PERIOD COVERED Semiannual Technical Summary 1 October 1980 – 31 March 1981
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Richard T. Lacoss		8. CONTRACT OR GRANT NUMBER*(s)* F19628-80-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order 3345 Program Element Nos. 61101E and 62708E Project Nos. 1D30 and 1T10
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE 31 March 1981
		13. NUMBER OF PAGES 76
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB Bedford, MA 01731		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

multiple-sensor surveillance system
multisite detection
target surveillance and tracking
communication network
acoustic sensors
low-flying aircraft
acoustic array processing
digital radio

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This Semiannual Technical summary reports work in the Distributed Sensor Networks program for the period 1 October 1980 through 31 March 1981. Status of the development and deployment of the DSN acoustic test bed and implementation of a real-time DSN capability using the test bed are reported. DSN communication requirements have been reviewed and a preliminary design has been completed for a communication channel structure and for protocols to provide the needed services. Simulation and analytical results have been obtained confirming the performance of the proposed distributed random-access scheme for local broadcast. Progress is reported on design studies directed toward an advanced nodal processor and software design and development efforts directed toward solving problems associated with distributed software systems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*